NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DESIGN AND IMPLEMENTATION
OF
INVENTORY DATABASE

by

Osman SARI

June 1985

Thesis Advisor: Samuel H. Parry

Approved for public release; distribution is unlimited

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* Design and Implementation of Inventory Database		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis June 1985 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Osman SARI		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93943-5100		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93943-5100		12. REPORT DATE June 1985 13. NUMBER OF PAGES 108
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* UNCLASSIFIED 15a. DECLASSIFICATION/ DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution is unlimited

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Database, Base Tables, Relations, Attribute

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

This thesis presents the design and implementation of the inventory database system. In order to effectively command and control the inventory of an Air Force, the commander must know the status of his resources. The use of a database management system can significantly increase his access to information regarding resource availability, location, state of operational readiness, and also increase end-user productivity, and decrease staff effort. The

(Continued)

ABSTRACT (Continued)

Semantic Data Model (SDM) was chosen as the method for designing the database. SDM provides an effective base for accommodating the evolution of the content structure and use of the database. After logical design of the inventory database, records are rearranged in order to satisfy relational database management system requirements. The inventory database is implemented by using the ORACLE relational DBMS.

Design and Implementation of Inventory Database

by

Osman SARI
Lieutenant Turkish Air Force
B.S., Turkish AIR War Academy, 1978

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1985

ABSTRACT (Continued)

Semantic Data Model (SDM) was chosen as the method for designing the database. SDM provides an effective base for accommodating the evolution of the content structure and use of the database. After logical design of the inventory database, records are rearranged in order to satisfy relational database management system requirements. The inventory database is implemented by using the ORACLE relational DBMS.

Design and Implementation of Inventory Database

by

Osman SARI
Lieutenant Turkish Air Force
B.S., Turkish AIR War Academy, 1978

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1985

ABSTRACT

This thesis presents the design and implementation of the inventory database system. In order to effectively command and control the inventory of an Air Force, the commander must know the status of his resources. The use of a database management system can significantly increase his access to information regarding resource availability, location, state of operational readiness, and also increase end-user productivity, and decrease staff effort. The Semantic Data Model (SDM) was chosen as the method for designing the database. SDM provides an effective base for accommodating the evclution of the content structure and use of the database. After logical design of the inventory database, records are rearranged in order to satisfy relational database management system requirements. The inventory database is implemented by using the ORACLE relational DBMS.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

My Ccuntry, Republic of Turkey, gave me a chance to study the graduate course of computer science in the Naval Postgraduate School. I am very grateful to many people for their help.

I would like to express my gratitude to my thesis advisor, Professor Samuel Parry and to my second advisor Dr. David K. HSIAO, for their enthusiastic guidance and support.

I am very thankful to my wife SENGUL, daughter ISIL, and son AKIN, for their understanding and encouragement during studying in the Naval Postgraduate School. It is time for me to work harder than before for my country.

I. INTRODUCTION

Databases are essential to an organization's information system. The information system supports the organization's functions, maintaining the data for these functions and assisting users to interpret the data for decision making. The database has a central role in this process. Database structures must be flexible to meet changing organizational needs. As new functions arise in an organization, new decisions follow in their wake. It should include facilities to allow the changes to be easily made. Characteristics of the database system will be discussed in the Chapter 2.

Meanwhile, it is not easy to develop database systems which perform in an optimal fashion. Different users will have different request about structuring data in the database. It is hard to satisfy all of the users with one type of structuring. There are different ways in which data can be structured. For that reason, in the database development phase all requests which come from users/organizations should be evaluated carefully by the designer(s).

For logical design of the inventory database the Semantic Data Model will be used. After that, the normal form concept of the Relational Database will be used to develop an inventory database.

Chapter 3 describes the basic concepts of database design which includes the logical and physical database design, and database models. Chapter 4 addresses the design of SDM and specifications of SDM. Chapter 5 describes how the inventory database is design by using the SDM. Chapter 6 addresses the basis structure of the relational model which includes functional dependency and normal form

concepts of the relational model. Chapter 7 describes the relational design criteria and relational design procedure. Also, in this chapter SDM for the inventory database will be transformed into a relational model. In Chapter 8, as a relational approach to database system, System R is described which contains architecture and system structure, the relational data system, and the relational storage system of System R. Chapter 9 describes the implementation of the inventory database by using ORACLE. Finally, Chapter 10 addresses the conclusions and recommendations of this thesis.

II. BASIC CONCEPT OF DATABASE

A. WHAT IS A DATABASE?

Database technology has been described as "one of the most rapidly growing areas of computer and information science." As a field, it is still comparatively young. Despite its youth, however, the field has quickly become one of considerable importance, both practical and theoretical. Today, many organizations have become critically dependent on the continued and successful operation of a database system.

Basically, a database is nothing more than a computer-based record keeping system: that is, a system whose overall purpose is to record and maintain information that may be necessary to the decision-making processes involved in management of that organization. In a database the data definitions and the relations between the data are separated from the procedural statements of a program. The question to be asked here is,"What is the major distinction between a database and a data file?" A database may have more than one use, and the multiple uses may satisfy multiple "views" of the data stored. A data file may have more than one use, but only one "view" of the stored data can be satisfied. Multiple views of a data file can be satisfied only after the data have been sorted. In a database environment, multiple uses may be the result of multiple users; for example, in a banking environment the information about customers may have several users, such as checking, savings, and installment loan. Thus data sharing is a major objective of an enterprise database system. A database system involves four major components: data,hardware, software, and users.

1. Data

A database is a repository for shared data. In general, it is both integrated and shared. " Integrated " means that the database may be thought of as a unification of several otherwise distinct data files, with any redundancy among those files partially or wholly eliminated. "Shared " means that individual pieces of data in the database may be shared among several users, in the sense that each of those users may have access to the same piece of data. Such sharing is really a consequence of the fact that the database is integrated. The term "shared" is frequently extended tc cover not only sharing as described above, but also concurrent sharing: that is, the ability of several different users to be actually accessing the database at the same time.

2. Hardware

The hardware consists of the secondary storage volumes - disks,drums,etc -on which the database resides, together with the associated devices, control units, channels, and so forth.

3. Software

Between the physical database itself (i.e, the data as actually stored) and the users of the system is a layer of software,usually called the database management system or DBMS. A database management system makes it possible to access integrated data that crosses operational, functional, or organizational boundaries within an enterprise. As an example of a Relaticnal DBMS System R will be evaluated in Chapter 8.

4. Users

Three classes of users can be considered. First, there is the application programmer, responsible for writing application programs that use the database, typically in a language such as COBCI or PL/I. The second class of user is the end-user, accessing the database from a terminal. An end-user can use a query language which as an integral part of the system. The third class of user is the database administrator, or DBA who is the person (or a group of persons) responsible for overall control of the database system.

B. OPERATIONAL DATA

Any enterprise such as a bank, hospital, university, or company must necessarily maintain large amounts of data about its operation, termed "operational data". The operational data for the enterprises would probably include account data, patient data, student data, product data, and planning data. Operational data does not include input or output data, work queues, or indeed any purely transient information. Input data refers to the information entering the system from the outside world; such information may cause a change to be made to the operational data but is not itself part of the database. Output data refers to messages and reports emanating from the system; such a report contains information derived from the operational data, but is not itself part of the database.

C. WHY DATABASE ?

The broad answer to this question is that a database system provides the enterprise with centralized control of its operational data. This is in sharp contrast to the

situation that prevails in many enterprises today where typically each application has its own files so that the operational data is widely dispersed, and therefore probably difficult to control. In the database system, the DBA has this central responsibility for the operational data. Some of the advantages that accrue from having centralized control of the data are described below.

1. Advantages of Database Systems

An important advantage of database processing is the elimination or reduction of data duplication. In nondatabase system each application has its own private files. This can often lead to considerable redundancy in stored data, with resultant waste in storage space. In the database, it need only be recorded once. Elimination of duplication saves file space and to some extent can reduce processing requirements. In some cases there may be some business reasons for maintaining multiple copies of the same data. In the database, however, redundancy should be controlled. The most serious problem of data duplication is that it can lead to a lack of data integrity. A common result of a lack of integrity is conflicting reports.

Data integration offers several important advantages. First and foremost, database processing enables more information to be produced from a given amount of data. Data are recorded facts or figures; information is knowledge gained by processing data.

Creation of program/data independence is another advantage of a database system. For the database application, application programs will obtain data from an intermediary, the DBMS. The application programs need not contain data structure, only the DBMS will need this structure. Another advantage of database processing is better data management. When data is centralized in a

database, one department can specialize in the maintenance of the data. That department can specify data standards and ensure that all data adhere to the standards.

Database processing creates another type of economy of scale. Since there is only one DBMS processing a shared database, improvements made to the database or to the DBMS will benefit many users.

2. Disadvantages of Database Systems

A major disadvantage of database processing is that it can be expensive. The DBMS may cost as much as $100,000 to buy. The database management system may occupy so much main memory that additional memory must be purchased. Even with more memory, it may monopolize the CPU, thus forcing the user to upgrade to a more powerful computer. Conversion from existing systems can be costly, especially if new data must be acquired.

Another major disadvantage is that the database processing tends to be complex. Large amounts of data in many different formats can be interrelated in the database. Both the database system and application programs must be able to process these structures, requiring more sophisticated programming. Backup and recovery are difficult in the database environment because of increased complexity and because databases are often processed by several users concurrently. Determining the exact state of the database at the time of the failure may be a problem. A final disadvantage is that integration, and hence centralization, increases vulnerability. A failure in one component of an integrated system can affect the entire system.

D. DATA INDEPENDENCE

In the conventional data set environment, the application programmer has to know answers to the following questions before manipulating the data :

1. What is its format?
2. Where is it located?
3. How is it accessed?

Changes in any of these three items may affect the application program and result in other changes, since the details of these three points may reside in the application code. The users of the database system should be oriented toward the information content of the data and should not be concerned with details of the representation and location. The ability to use the database without knowing the representation details is called DATA INDEPENDENCE. Data independence provides that the individual application programmer no longer must change the application programs to accommodate changes in access method or location or format of the data. The reasons for data independence are as follows:

1. To allow the DBA to make changes in the content, location, representation and organization of a database without causing reprogramming of application programs which use the database.
2. To allow the supplier of data processing equipment and software to introduce new technologies without causing reprogramming of the customer's application.
3. To facilitate data sharing by allowing the same data to appear to be organized differently for different application programs.
4. To simplify application program development and,in particular, to facilitate the development of programs for interactive database processing.

5. To provide the centralization of control needed by the DBA to insure the security and integrity of the database.

E. DATA DICTIONARY

A data dictionary is a central repository of information about the entities, the data elements representing the entities, the relationships between the entities, their origins, meanings, uses, and representation formats. A facility that provides uniform and central information about all the data resources is called a DATA DICTIONARY (DD). The benefits of using a data dictionary are related to the effective collection, specification, and management of the total data resources of an enterprise. A data dictionary should help a database user in:

1. Communicating with other users.
2. Controlling the data elements in a simple and effective manner; that is, introducing new elements into the system, or changing the descriptions of the elements.
3. Reducing the data redundancy and inconsistency.
4. Determining the impact of the changes to the data elements on the total database.
5. Centralizing the control of the data elements as an aid in database design and in expanding the design.

III. DATABASE DESIGN

A database is the interface between people and machines. The nature of these components is very different. The difficulty is to develop a database design which meets the needs of the people who will use it and which is practical in term of technology and hardware. Since the database is the bridge between humans on one side and hardware on the other, it must match the characteristics of each.

There is no algorithm for database design. Database design is both art and science. Dealing with people, understanding what they want today, predicting what they will want tomorrow, differentiating between individual needs and community needs, and making appropriate design tradeoffs are artistic tasks. There are principles and tools, but these must be used in conjunction with intuition and guided by experience.

Database design is a two-phased process. The first phase of the database design is usually called the Logical Database Phase in which the designer examines the users' requirements and builds a conceptual database structure that is a model of the organization. Once the logical design of the database is completed, this design is formulated in terms of a particular DBMS. Usually compromises must be made. The process of formulating the logical design in terms of a DBMS facility is called Physical Database Design. This chapter considers both phases of the database design.

A. LOGICAL DATABASE DESIGN

Typically, database design is an iterative process; during each iteration, the goal is to get closer to an

acceptable design. Thus a design will be developed and then reviewed. Defects in the design will be identified, and the design will be revised. This process is repeated until the development team and users can find no major defects. This does not mean the design will work; it simply means no one can think of any reason why it will not work. Figure 3.1 illustrates the steps in a typical database design project[Ref.4].

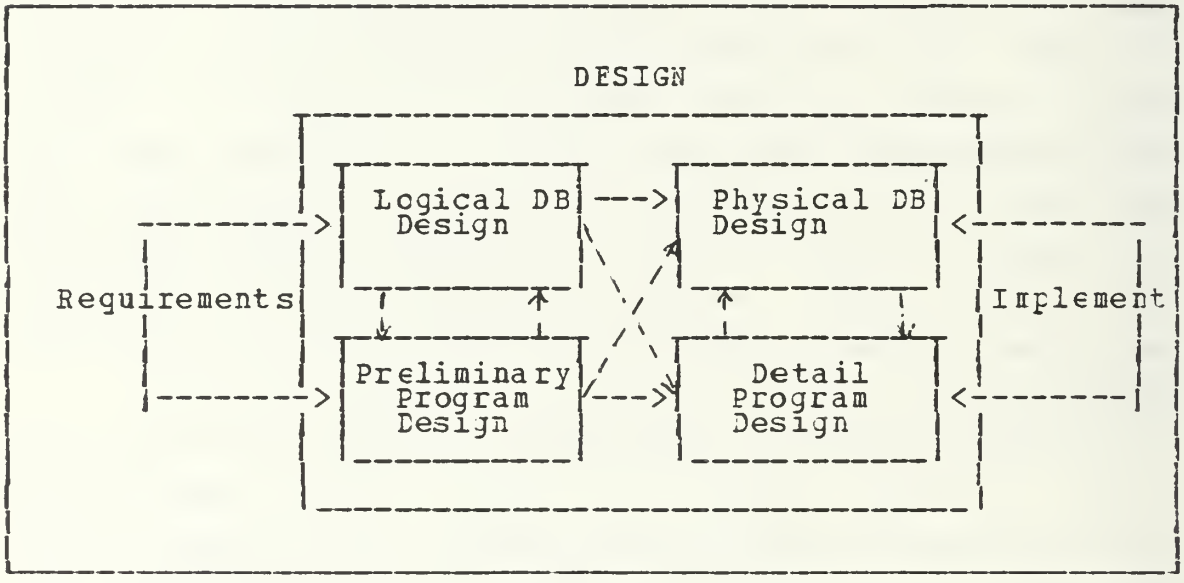

Figure 3.1 Database and Program Design Flow.

User requirements are studied and a logical database design is developed. Concurrently, the preliminary design of the database processing programs is produced. Next, the logical database and the preliminary program designs are used to develop the physical database design and the detail program design specifications. Finally, both of these are input to the implementation phase of the project.

1. Inputs to Logical Database Design

The inputs to the logical database design are the system requirements and the project plan. Requirements are determined by interviews with users, and then they are approved by both users and management. The project plan describes the system environment, the development plan, and constraints and limitations on the system design. Policy statements can be used to develop the descriptions of the logical database design.

2. Outputs of the Logical Database Design

A logical database design specifies the logical format of the database. The records to be maintained, their contents, and relationships among these records are specified.

To specify logical records, the designer must specify the levels of the detail of the database model. If the model is highly aggregated and generalized, there will be few records. If the model is detailed, there will be many records. The designer must examine the requirements to determine how coarse or how fine the database model should be. The contents of these are specified during logical design. Names of fields and their formats must be determined. As the requirements are evaluated and the design progresses, constraints on data items will be identified. These are limitations on the values that data can have. These types of constraints are common. Field constraints limit the values that a given data item can have. Intrarecord constraints limit values between fields within a given record. Also, record relationships are specified during logical design. The designer studies the application environment, examines the requirements, and identifies necessary relationships. Finally, output of the logical

database design is the specification of the database records, their contents, constraints, and relationships.

3. Stages of Logical Database Design

Many techniques have been developed for logical database design. Some techniques are completely intuitive and others involve specific procedures for processing a data dictionary. Others are between these extremes. The major steps in the logical database design are as follows.

a. Identify Data to be Stored

First, the data dictionary is processed and data that is to be stored is identified and segregated. This step is necessary because the data dictionary will contain the description of the reports, screens, and input documents that will not be part of the database.

b. Consolidate and Clarify Data Names

The next step is to clarify the terms used for the data. One task is to identify synonyms, to decide on standard names for synonyms, and the record aliases. Synonyms are two or more names for the same data item. They arise because of the terminology differences within the organization. In this case the designer will need to select a single , standard name for the data item in the logical schema of the database. In some cases synonyms can not be eliminated because the users want to maintain their own terminology.

Another task related to terminology is to ensure that data items having the same name are truly the same. If not, unique data item names must be developed. Consider the data item DATE. This can be the date of shipment, the date of employee termination, or the date of order. The designer must determine if all of the uses of the DATE item are the same. If not, new and unique names must be determined.

c. Develop the Logical Schema

The third step in the design process is to develcp the logical schema by defining records and relationships. Records are defined by determining the data items they will contain. The designers examine the data flow diagrams and data dictionary, apply intuition to the business setting of the new system, and determine that certain records will need to exist. After this determination, some of the files may need to be combined and some of them may not.

The second step in developing the logical schema is to determine the relationships among database records. At that point, representation of the relationships by the database system is nct important. Instead, the design team wants to model how the users see the relationships. We do not need to consider physical limitations at this point. Doing so makes the logical schema too complex and may constrain our thinking so that we miss good design alternatives. At that point, the design team must discriminate between theoretical and useful relationships. A theoretical relationship can exist logically, but never be needed in practice. In general, if there is any question regarding whether a relationship is useful or not, then the relationship should be included in the logical schema. The relationship always can be omitted later in the physical design, whereas if the relationship were omitted during logical design, it would be difficult to add later.

d. Define Processing

The next step is to define the processing cf the database. The requirements are examined to determine hcw the database should be manipulated to produce required results. The processing definitions can be developed in several ways.

One method is to describe transactions and data to be modified. Another method is to develop structure charts of the programs that will access the database. This process is important because concurrent design of the programs and database will improve the database design. It is also clear that concurrent design improves the quality of programs.

e. Design Review

The final stage of the logical database design is a review. The logical schema and users' views are examined in the light of the requirements and program descriptions. Every attempt is made to identify omissions and unworkable aspects of the design. Typically, a panel of independent data processing people is convened for this review. Documentation of the logical schema, users' views, and program descriptions are examined by the panel, and oral presentations are evaluated.

At the conclusion of the design review, the panel produces a list of problems discovered and a recommendation regarding the next step to be taken.

B. PHYSICAL DATABASE DESIGN

The second stage of the database design is physical design which is a stage of the transformation. The logical schema is transformed into the particular data constructs that are available with the DBMS to be used. As mentioned before, the inputs to the physical database design are the outputs of the logical database design, the system requirements, and the preliminary design of programs. Whereas the logical design is DBMS independent, the physical design is very much IBMS dependent. Detail specification of the database structure is produced. These specifications will be used during database implementation to write source

statements that define the database structure to the DBMS. These statements will be compiled by the DBMS and the object form of the database structure will be stored within the database as shown in Figure 3.2 [Ref.4].

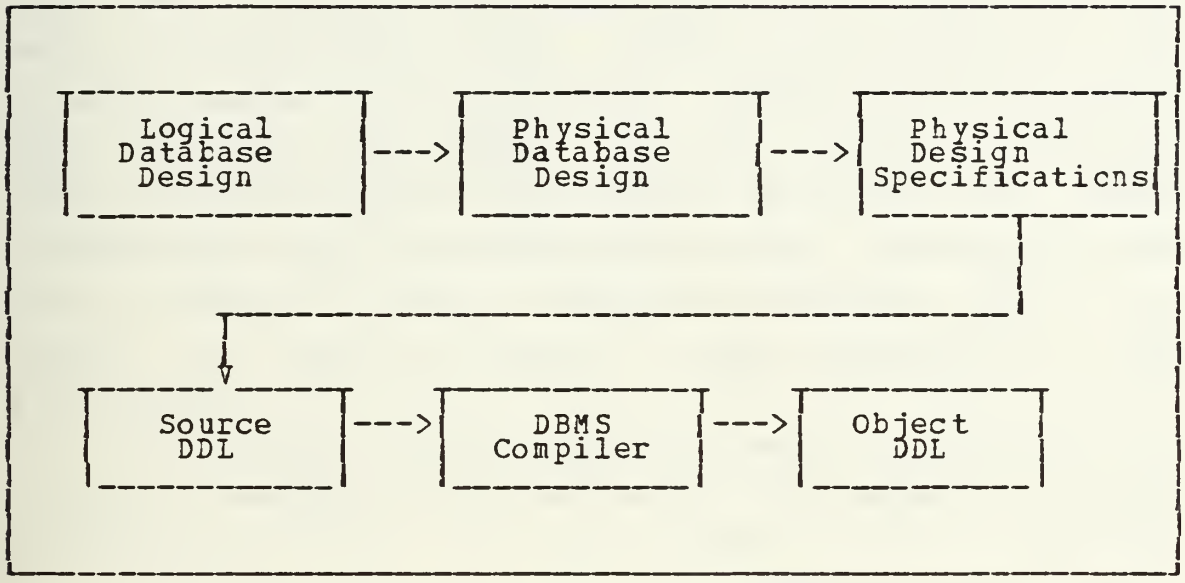

Figure 3.2 Physical Design Process.

1. Physical Design Steps

Practical experience has shown that neither the starting point nor the order of steps can be definitely stated for a given design problem. On the other hand, the physical design phase can be regarded as an iterative process of initial design and requirement. Each step needs to be performed several times, but succeeding analysis should be done more quickly because the procedure is known and the number of unchanging performance variables should increase between iterations. Steps of physical design are as follows.

a. Stored Record Format Design

Assuming that the logical record structure has been defined, this process addresses the problem of formatting stored data by analysis of the characteristics of data item types, distribution of their values, and their usage by various applications. Certain data items are often accessed more frequently than others, but each time a particular piece of data is needed, the entire stored record, and all stored records in a physical block as well, must be accessed. Record partitioning defines an allocation of individual data items to separate physical devices of the same or different types, or separate extents on the same device, so that the total cost of accessing data for a given set of user applications is minimized. Logically, data items related to a single entity are still considered to be connected, and physically they can still be retrieved together when necessary.

2. Stored Record Clustering

Record clustering refers to the allocation of the records of different types into physical clusters to take advantage of physical sequentiality whenever possible. Associated with both record clustering and record partitioning is the selection of physical block size. Blocks in a given clustered extent are influenced somewhat by stored record size, but also by storage characteristics of physical devices. Choice of block size may be subject to considerable revision during an iterative design process.

3. Access Method Design

An access method provides storage and retrieval capabilities for data stored on physical devices, usually secondary storage. The two critical components of an access

method are storage structure and search mechanism. Storage structure defines the limits of possible access paths through indexes and stored records, and search mechanisms define which paths are to be taken for given applications. Access method design is often defined in terms of primary and secondary access path structure. The primary access paths are associated with initial record loading, or placement, and usually involve retrieval via the primary key. Secondary access paths include interfile linkages and alternate entry-point access to stored records via indexes and secondary keys. The trade-off is that access time can be greatly reduced through secondary indexes, but at the expense of increased storage space overhead and index maintenance.

A fourth step of physical design trade-offs among integrity, security, and efficiency requirements also should be considered.

a. Program Design

The goal of the physical data independence, if met, produces application program modification due to physical structure design decisions. Standard DBMS routines should be used for all accessing, and query or update transaction optimization should be performed at the system software level. Then, application program design should be completed when the logical database structure is known. When physical data independence is not guaranteed, program modification is likely.

4. Physical Design Environment

The design environment is basically the same for both file design and physical database design. Major categories of inputs and outputs for the physical design phase are illustrated in Figure 3.3. The logical database

structure resulting from the implementation design phase defines the framework from which the physical designer works. If no catastrophic inefficiency is detected, it will remain unchanged during physical design. In general, new parameters will be considered, but previous tentative decisions on access paths and record allocation are finalized in this phase. New parameters are those specific to DBMS and operating system access methods, those specific to describe physical device capacity limitations and timing characteristics, and all operational requirements which are constraints imposed on integrity, security, and response time under static conditions and for dynamic growth projections. During the design process, consideration of efficiency issues can take place only the after various constraints are satisfied and a feasible solution has been obtained.

5. Performance Measures

The determination of performance measures for physical design is most critical to the design process. They affect not only the design choices, but also the techniques employed to determine those choices.

Multiple performance measures provide the designer with flexibility for decision making for both the initial design procedure and for future modifications. If we describe the database system performance in terms of cost we should consider life cycle cost in terms of following items:

1. Planning cost
2. Design cost: programs, databases
3. Implementation and testing cost:programs, databases
4. Operational cost:users, computer resources
5. Maintenance cost:program errors, data integrity loss.

The major problem that the physical database designer must address is how to minimize present and future

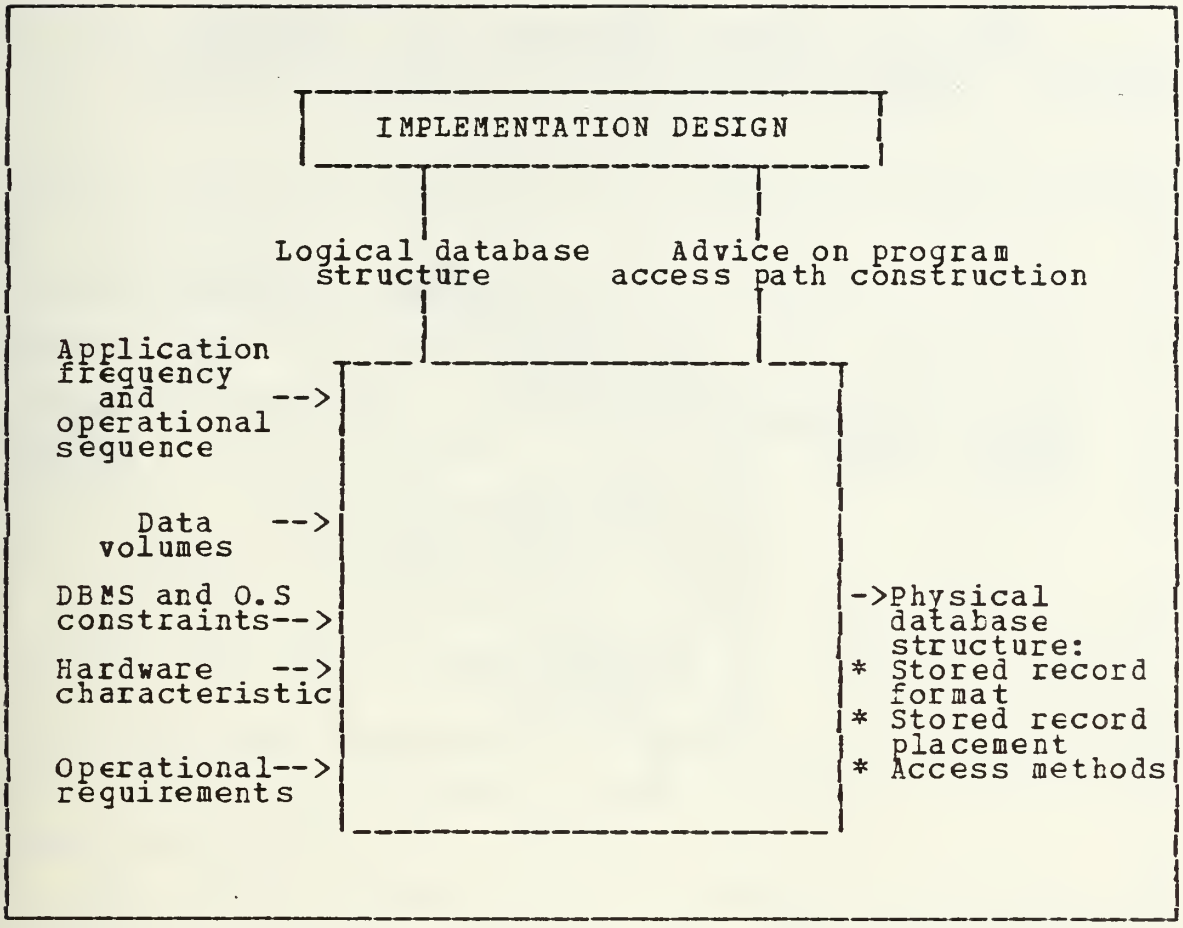

Figure 3.3 Physical Design Environment.

operational costs in terms of user needs and computer resources. The remainder of the life cycle phases' costs are well defined for general software systems. Operational costs are unique to physical design and can be categorized as follows:

1. Query response time
2. Update transaction cost
3. Report generation cost
4. Reorganization frequency and cost
5. Main storage cost
6. Secondary storage cost

Each of these components is important to the designer; typical considerations are shown in Figure 3.4.

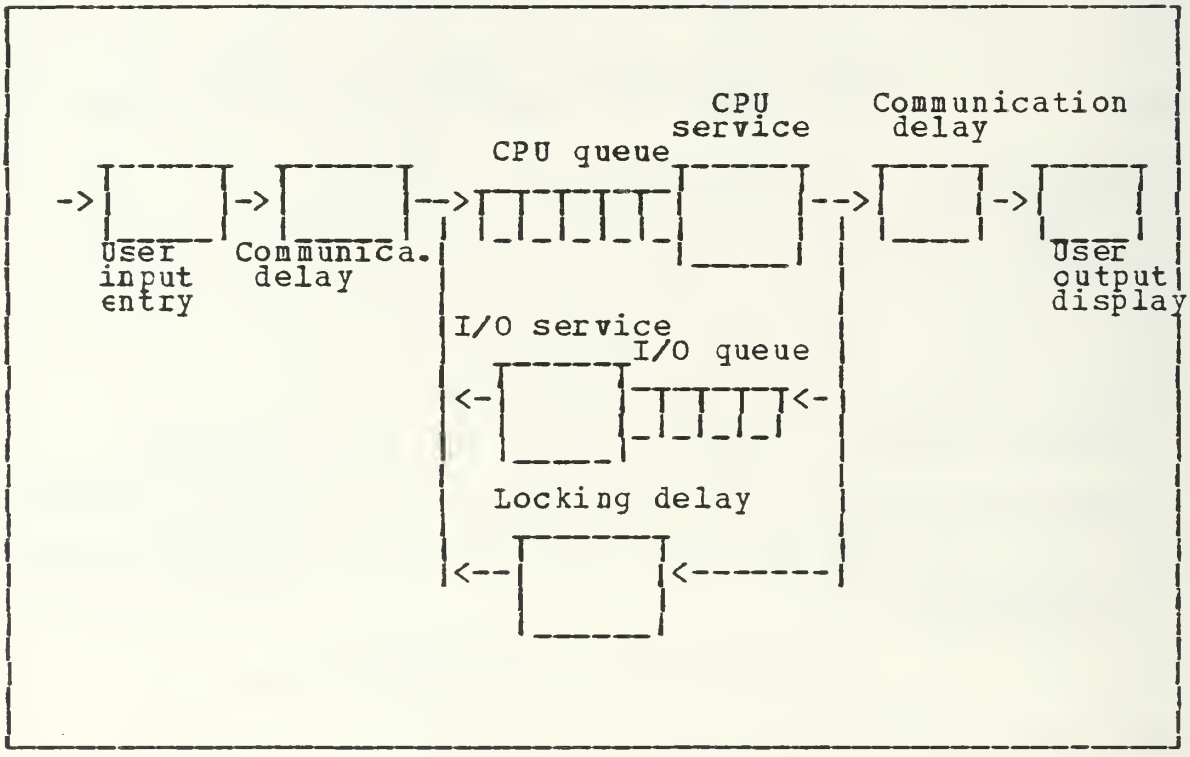

Figure 3.4 Query Response Time Components.

C. DATABASE MODELS

A database model is vocabulary for describing the structure and processing of the database. There are two reasons for studying database models. First, they are important database design tools. Database models can be used for both logical and physical database design - much as flowcharts or pseudocode are used for programs design. Second, database models are used to categorize DBMS products. Database models have two major components. First, the data definition language (DDL)is a vocabulary for

defining the structure of a database. The DDL must include terms for defining records, fields, keys, and relationships. Also it should provide a facility for expressing a variety of user views. As a second component, data manipulation language (DML) is a vccabulary for describing the processing of the database. Two types of DML exist: procedural and nonprccedural DML. Facilities are needed to retrieve and change data for both. Procedural DML is language for describing actions to be performed on the database. It obtains a desired result by specifying the operations to be performed. Nonprocedural DML is language for describing the data that is wanted without describing how to obtain it.

Figure 3.5 illustrates six common and useful database models. The models are arranged on a continuum. Models on the left-hand side cf this figure tend to be oriented to humans and human meaning, whereas those on the right-hand side are more oriented toward machines and machine specifications[Ref.4: p. 215].

The primary purpcse of this thesis is to design and implement an inventory database. For the logical design of this database, the Semantic Data Model (SDM) will be used and for physical design a Relational Model will be employed. For this reason, SDM and the Relational model will be discussed in detail in the following chapters.

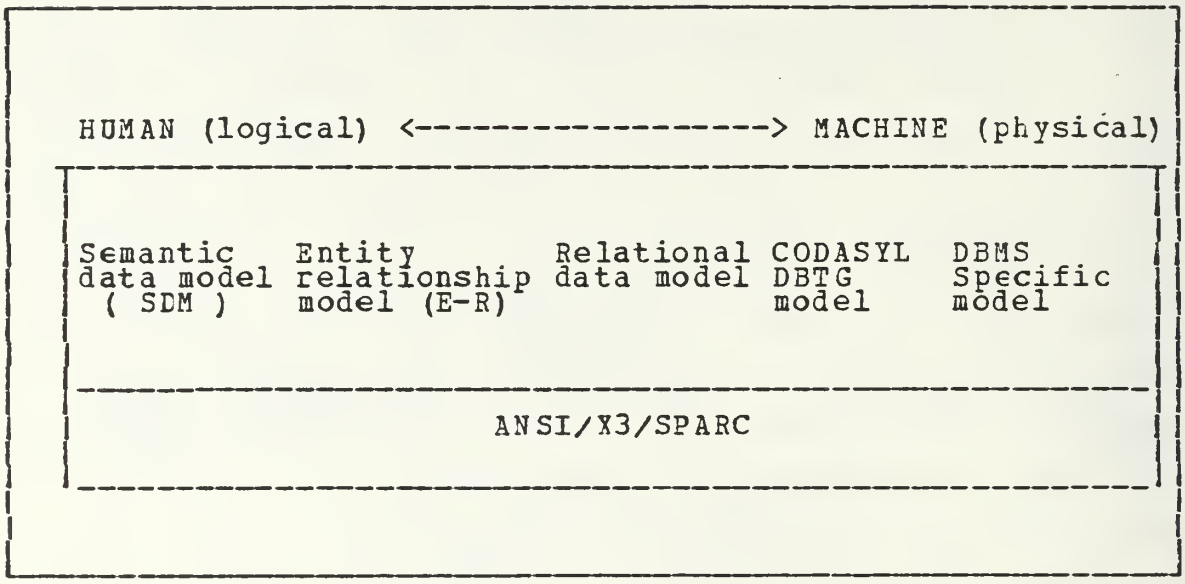

Figure 3.5 Relaticnships of Six Important Data Model.

IV. SEMANTIC DATA MODEL (SDM)

A. INTRODUCTION

The Semantic Data Model was developed by M. HAMMER and D. McLOED in 1981[Ref.13]. SDM is a high-level semantics-based database description and structuring formalism for databases. This database model is designed to capture more of the meaning of an application environment than is possible with contemporary database models. An SDM specification describes a database in terms of the kinds of entities that exist in the application environment, the classifications and groupings of those entities, and the structural interconnections among them. SDM provides a collection of high-level modeling primitives to capture the semantics of an application environment. SDM is designed to enhance the effectiveness and usability of database systems. An SDM database description can serve as a formal specification and documentation tool for a database.

Every database is a model of some real world system. The contents of a database are intended to represent a snapshot of the state of an application environment and each change to the database should reflect an event occuring in that environment. It is appropriate that the structure of the database mirror the structure of the system that is being modelled[Ref.13]. A database whose organization is based on naturally occuring structures will be easier for a database designer to construct and modify than one that forces him to translate the primitives of his problem domain into an artificial specification construct. Similarly, a database user should find it easier to understand and employ a database if it can be described to him using concepts with which he is already familiar.

Contemporary database models provide the data structures which do not adequately support the design, evolution, and use of complex databases. These models have significantly limited capabilities for expressing the meaning of a database. The semantics of a database defined in terms of these mechanisms are not readily apparent from the schema which is the global user view of a database; instead, the semantics must be separately specified by the database designer and consciously applied by the user.

B. THE DESIGN OF SDM

SDM has been defined with a number of specific kinds of uses in mind. First, SDM is meant to serve as a formal specification mechanism for describing the meaning of a database; an SDM schema provides a precise documentation and communication medium for database users. For a new user of a complex database, it is easy to find out what information is contained in the database. Second, SDM provides the basis for a variety of high-level semantics-based user interfaces to a database; these information facilities can be constructed as front-ends to existing database management systems, or as the query language of a new DBMS. Such interfaces improve the process of identifying and retrieving relevant information from the database. Finally, SDM provides a foundation for supporting the effective and structured design of databases and database intensive application systems.

SDM has been designed to satisfy a number of criteria that are not met by contemporary database models, but are essential in an effective database description and structuring formalism. They are as follows[Ref.13]:

> "The constructs of database model should provide for the explicit specification of a large portion of the meaning of a database. Many contemporary database models (such

as the CODASYL DBTG network model and the hierarchical model) exhibit compromises between the desire to provide a user-oriented database organization and the need to support efficient database storage and manipulation facilities. By contrast, the relational database model stresses the separation of user-level database specifications and underlying implementation detail(data independence). However, the semantic expressiveness of the hierarchical model, network and relational models are limited; they do not provide sufficient mechanisms. to allow a database schema to describe the meaning of a database. They employ overly simple data structures to model an application environment. In so doing, they inevitably lose information about the database. This is a consequence of the fact that their structures are essentially all record-oriented constructs; the appropriateness and adequacy of the record construct for expressing database semantics is highly limited. It is essential that the database model provide a rich set of features to allow the direct modeling of application environment semantics. A database model must support a relativest view of the meaning of a database , and allow the structure of a database to support alternative ways of looking at the same information. Flexibility is essential in order to allow for multiple and coequal views of the data. In a logically redundant database schema, the values of some database components can be algorithmically derived from others. Incorporating such derived information into a schema can simplify the user's manipulation of a database by statically embedding in the schema data values that would otherwise have to be dynamically and repeatedly computed. Finally, an integrated schema explicitly describes the relationships and similarities between multiple ways of viewing the same information. Contemporary database models do not adequately support relativism. In these models, it is generally necessary to impose a single structural organization of the data, one which inevitably carries along with it a particular interpretation of the data's meaning."

A database model must support the definition of schemata that are based on abstract entities. Specifically, this means that a database model must facilitate the description of relevant entities in the application environment, collections of such entities, relationships among entities, and structural interconnections among the collections. Moreover, the entities themselves must be distinguished from their syntactic identifiers; the user-level view of database should be based on actual entities rather than on artificial entity names. Allowing entities to represent themselves makes it possible to directly reference an entity from a related one. In record-oriented database models, it is

necessary to cross reference between related entities by means of their identifiers[Ref.11].

C. A SPECIFICATION OF SDM

The following general principles are specified by McLoed and Hammer in 1981:

1. A database is to be viewed as collections of entities that correspond to the actual objects in the application environment.
2. The entities in a database are organized into classes that are meaningful collections of entities.
3. The classes of a database are not generally independent, but rather are logically related by means of interclass connections.
4. Database entities and classes have attributes that describe their characteristics and relate them to other database entities. An attribute value may be derived from other values in the database.
5. There are several primitives for defining interclass connections and derived attributes, corresponding to the most common types of information redundancy appearing in database applications. These facilities integrate multiple ways of viewing the same basic information, and provide building blocks for describing complex attributes and interclass relationships.

1. Classes

As mentioned above, an SDM database is a collection of entities that are organized into classes. Figure 4.1 shows a basic format of an SDM entity class description. The structure and organization of an SDM database is specified

by an SDM schema, which identifies the classes in the database. Each class in an SDM schema has the following features.

1. A class name identifies the class. Multiple synonymous names are also permitted. Each class name must be unique with respect to all class names used in the schema.
2. The class has a collection of members: the entities that constitute it. Each class in an SDM schema is a hcmogeneous cclllection of one type of entity at an appropriate level of abstraction. The entities in a class may correspond to various kinds of objects in the applicatiцn environment.
3. An optional textual class description describes the meaning and ccntent of the class. A class description should be used to describe the specific nature of entities that constitute a class and to indicate their significance and role in the application environment.
4. The class has a collection of attributes that describes the members of that class or the class as a whole. There are two types of attributes, classified according to applicability.
5. A member attribute describes an aspect of each member of a class by logically connecting the member to one or more related entities in the same cr other classes. A class attribute describes a property of a class taken as whole.
6. The class is either a base class or a nonbase class. A base class is one that is defined independently of all other classes in the database; it can be thought of as modeling a primitive entity in the application environment. Base classes are mutually disjoint in that every entity is a member of exactly one base

class. A nonbase class is one that does not have independent existence; rather, it is defined in terms of one or more other classes. In SDM, classes are structurally related by means of interclass connections. Each nonbase class has associated with it an interclass connection. If class is base class, it has an associated list of groups of member attributes; each of these groups serves as a logical key to uniquely identify the members of a class. If the class is base class, it is specified as either containing duplicates or not containing duplicates.

a. Interclass Connections

There are two main types of interclass connections in SDM: the first allows subclasses to be defined and the second supports grouping classes. The subclass connection specifies that the members of nonbase class (S) are of the same basic entity type as those in the class to which it is related via interclass connection. This type of interclass connection is used to define a subclass of a given class. A subclass S of class C is a class that contains some, but not necessarily all, of the members of C. In SDM, a subclass S is defined by specifying a class C and a predicate P on the member of C; S consists of just those members of C that satisfy P. Several types of predicates are permissible. A predicate on the member attributes of C can be used to indicate which members of C are also members of S. The predicate "where specified" can be used to define S. This means that S contains at all times only entities that are members of C. It is also possible to define subclass S as an intersection of database classes (C1,C2).

The other type of interclass connection allows for the definition of nonbase class, called a grouping class (G), whose members are of a higher-order entity type than

```
ENTITY_CLASS_NAME
     [description: ..........]
     [interclass connection: .........]
      member attributes:
           Attribute_name
                 value class: ..........
                [Mandatory]
                [multivalued][no overlap in values]
                [exhaust value class]
                [not changeable]
                [inverse: Attribute_name]
                [match: Attribute_name of ENTITY_CLASS
                        on Attribute_name2]
                [derivation: ............]
      [ class attributes:
            Attribute_name
                 [Description: ......]
                   value class: .....
                 [derivation: .......]]
      [identifiers:
            Attribute_name1+[Attribute_name2+[..]]]
```

Figure 4.1 Format of SDM Entity Class Description.

those in the underlying class (U). A grouping class is second order, in the the sense that its members can themselves be viewed as classes; in particular, they are classes whose members are taken from U.

2. Attributes

Each class has an associated collection of attributes. Each attribute has the following features.

1. An attribute name identifies the attribute. An attribute name must be unique with respect to the set of all attribute names used in class, the class's underlying base class, and all eventual subclasses of that base class.
2. The attribute has a value which is either an entity in the database or a collection of such entities. The value of an attribute is selected from its underlying value class, which contains the permissible values of the attribute.
3. The attribute is either a member attribute which applies to each member of the class, and so has a value for each member, or a class attribute which applies to a classes a whole, and has only one value for the class.
4. The attribute is specified as either single valued or multivalued. The value of a single-valued attribute is a member of the value class of the attribute. The value of a multivalued attribute is a subclass of the value class. Thus, a multivalued attribute itself defines a class, that is, a collection of entities. A multivalued member attribute can be specified as nonoverlaping which means that the values of the attribute for two different entities have no entities in common; that is, each member of the value class of the attribute is used at most once.
5. An attribute can be specified as mandatory, which means that a null value is not allowed for it.
6. An attribute can be specified as not changeable which means that once set to a nonnull value, this value

cannot be altered except to correct an error. pcintend

a. Member Attribute Interrelationships

(1) Inverse. The first way in which a pair of member attributes can be related is by means of inversion. Member attribute X1 of class Y1 can be specified as the inverse of member X2 of Y2 which means that the value of X1 for a member M1 of Y1 consists of those members of Y2 whose value of X2 is M1. The inversion interattribute relationship is specified symmetrically in that both an attribute and its inverse contain a description of the inversion relationship. A pair of inverse attributes establish a binary association between the members of the classes that the attributes modify.

(2) Matching. The second way in which a member attribute can be related to other information in the database is by matching the value of the attribute with some member(s) of a specified class. The value of match attribute A1 for the member M1 of class C1 is determined as follows.

1. A member M2 of some class C2 is found that has M1 as its value of member attribute A2.
2. The value of member attribute A3 for M2 is used as the value of A1 for M1.

If A1 is a multivalued attribute, then it is permissible for each member of C1 to match the members of C2; in this case, the collection of A3 values is the value of attribute A1.

Matching permits the specification of binary and higher degree associations, while inversion permits the binary associations. The combined use of

inversion and matching allows an SDM schema to accommodate relative viewpoints of an association.

(3) Derivation. SDM provides the ability to define an attribute whose value is calculated from other information in the database. Such an attribute is called derived, and the specification of its computation is its associated derivation. The following rules are formulated by HAMMER and McLOED, in order to allow the use of derivations while avoiding the danger of inconsistent attribute specifications.

1. Every attribute may or may not have an inverse; if it does, the inverse must be defined consistently with the attribute.
2. Every member attribute A1 satisfies one of the following cases:

1. A1 has exactly one derivation. In this case, the value A1 is completely specified by the derivation. The inverse of A1, if it exists, may not have a derivation or matching specification.
2. A1 has exactly one matching specification. In this case, the value of A1 is completely specified by its relationships with an entity to which it is matched. The inverse of A1, if it exists, may not have a derivation.
3. A1 has either a matching specification or a derivation. In this case, it may be that the inverse of A1 has a matching specification or a derivation; if so, then one of the above two rules applies.

D. ADVANTAGES OF SDM

1. SDM provides an effective base for accommodating the evolution of the content structure and use of a database. Relativism, logical redundancy, and derived

information support this natural evolution of schemata.

2. SDM supports a basic methodology that can guide the Database Administrator (DBA) in the design process by providing him with a set of natural design templates. The DBA can approach the application in question with the intent of identifying its classes, subclasses, and so on. Then he can select representations for these constructs.
3. It provides a facility for expressing meaning about the data in the database. During logical database design, the designer needs such a facility to avoid confusion and to document learning, design decisions, and constraints. SDM provides better facilities for such documentation than other data models.
4. It allows data to be described in context. Users see data from different perspectives.
5. In SDM, constraints on operational data can be defined. For example, if a given item is not changeable, SDM allows this fact to be stated.
6. An SDM schema for a database can serve as a readable description of its contents, organized in terms that a user is likely to be able to comprehend and identify.

V. SEMANTIC DESIGN OF INVENTORY DATABASE

Figures 5.2, 5.3, 5.4, 5.5, 5.6 and 5.7 describe the logical schema of the inventory database. There are five records in the logical schema. IDENTIFICATION record gives all the information about a given item in the Air Force inventory such as national stock number, document which provides technical information about the item, total quantity in the inventory, total amount used in the past, maximum authorized quantity to keep in the inventory, who is authorized to use, depot in which item is stocked, total number of the item used by units, supplier name who supplies item, and amount purchased in the past. The second record is UNIT which provides information about units in which an item is used. It has several fields such as unit code, superior command, national stock number of item which is used in the unit, quantity on hand, used amount, required amount, location of unit and subordinate command. The third record is the ORDER. This record describes the ordering process of the item. Supplier name, Nsn_no, date, amount and shipment type are the member attributes of the ORDER. The fourth record is DEPOT_STOCK_LEVEL which provides data about stock status of the item. Its fields are depo_id, Nsn_no_registered, stock_amount, and supplier name. The SUPPLIER record provides data about suppliers who supply the items to the Air Force. Supplier name, country, city and address are the member attributes of the SUPPLIER.

In the logical schema of the inventory database all classes and their member attributes are informally defined and special remarks are written. The purpose of this process is to present the semantic of the database which will let the user easily understand the database. Figure 5.1 shows

the general structure of the records and the member attribute interrelationships. As mentioned in the previous chapter SDM provides three facilities for defining relationships. All three facilities use the SDM characteristic that entities can be contained within entities. Derivation, inverse, and match facilities are discussed in Chapter 5.

In the IDENTIFICATION record there is a derivation between Tot_used_in_past and and Sum_of_used_Units. This means that Tot_used_in_past is derived from Sum_of-used_Units by summation as specified. Also there is match between past_amount_purchased of IDENTIFICATION class and amount of ORDER class. This means that when the order occured, the value of this member will move the past_amount_purchased of IDENTIFICATION. On the class level, a member of IDENTIFICATION is to be matched with a member of ORDER. This is physically meaningful as well as logically. When the logistic department ordered an item and receives this order, this value should be moved to the past_amount_purchased in order to keep the correct data. For this reason, the member in the IDENTIFICATION class must match the value in the amount of the ORDER. Otherwise there can be an inconsistency in the database.

There are three inverse relationships in the logical schema. First, between aut_to_use of IDENTIFICATION class and Nsn_no_use of UNIT class, secondly between depot_of-registry of IDENTIFICATION class and Nsn_no_registered of DEPOT_STOCK_LEVEL class, and third between superior_comm of UNIT class and subordinate_comm of UNIT class. The logic is the same for all. The inverse facility causes two entities to be contained with each other. As [Ref.4] specified, this is physically impossible, so this idea may seem a bit strange. Consider the first inverse. The attributes of IDENTIFICATION and UNIT are

inverses of each other. In IDENTIFICATION, the attribute aut_to_use has the value class UNIT and the inverse attribute Nsn_no_use. In the UNIT, the attribute Nsn_no_use has the value class IDENTIFICATION and inverse attribute aut_to_use. As menticned in the previous chapter, inverses are always specified by such pairs. In the second inverse, depot_of _registry cf IDENTIFICATION has the value class DEPOT_STOCK_LEVEL and inverse attribute Nsn_no_registry; Nsn_nc_registry of DEPOT_STOCK_LEVEL has the value class IDENTIFICATION and inverse attribute depot_of_registry.

It is also possible in the SDM to define an inverse relationship between two attributes which are in the same class. This case occurrs in UNIT class. Superior_comm and subordinate-comm are the inverse of each other. Both of them have the same value class, UNIT. Here, the inverse interattribute relationship is specified symmetrically. Supericr_command ccmmands the subordinate_command and subordinate_command is commanded by the superior_command. Users can describe the data in a manner which fits their logical view.

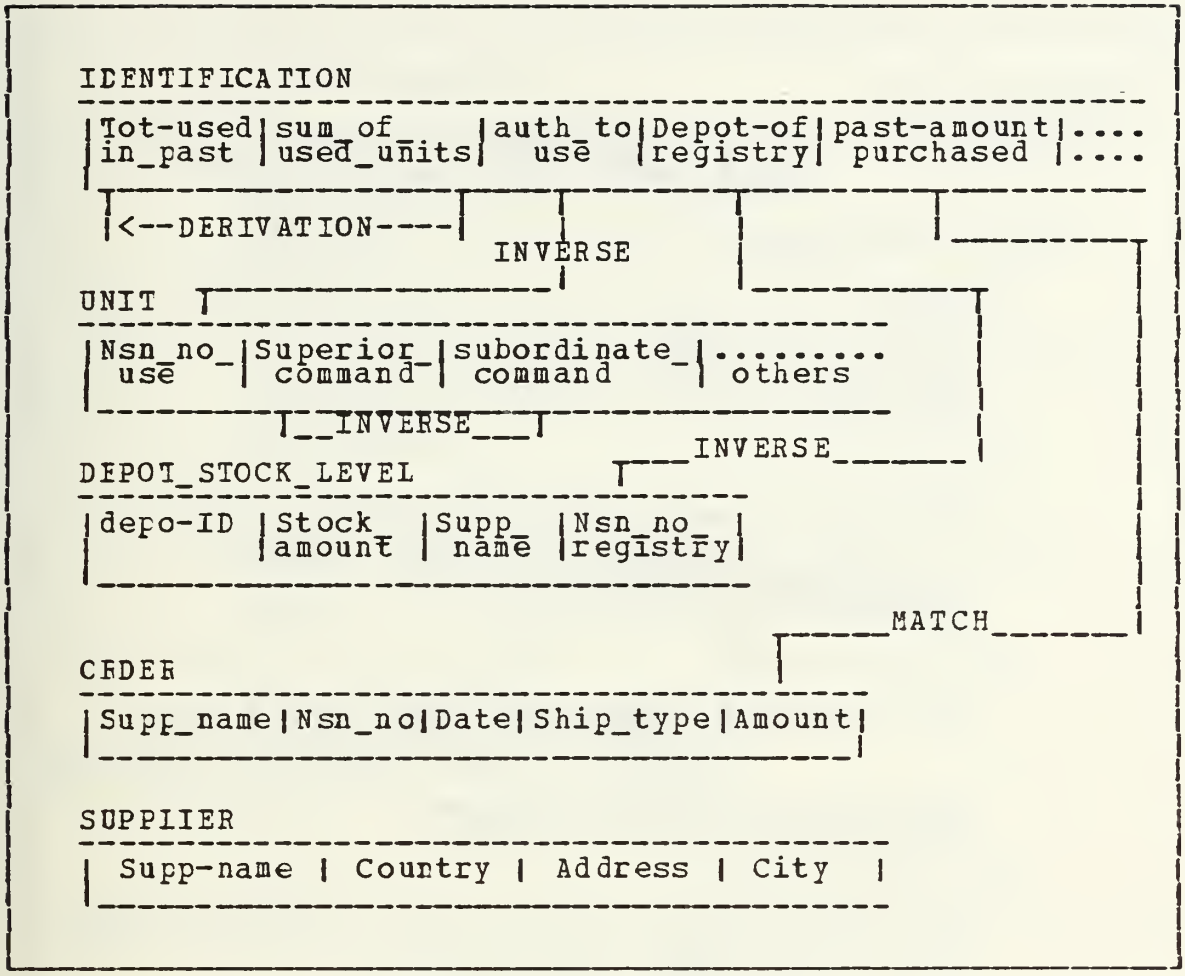

Figure 5.1 Interclass Relationships of SDM Design.

```
IDENTIFICATION

   Description : Overall information about a given
                 item which is in the Air Force
                 inventory

   Member attributes:

      Nsn_no
        description:National stock number of a given
                    item.
        value class:NATIONAL_STOCK_NUMBER
        mandatory
        not changeable

      Document

        description:Technical Order[TO] for a given
                    item. It specifies technical
                    information about item(s).
        value class:DOCUMENTATION
        mandatory

      Tot_Qty_on_Hand

        description:It specifies quantity which is
                    currently available for a given
                    item in the Air Force (AF) logistics
                    system.
        value class:QUANTITY_ON_HAND
        mandatory

      Tot_Used_In_Past

        description:Total amount which is used in the
                    past.
        value class:TOTAL_USED_IN_PAST
        derivation :Sum of used_UNITS

      Max_Auth_Qty_On_Hand

        Description:Maximum number of items that AF
                    logistics department authorized to
                    hold, not more than this capacity.
        value class:MAX_AUTH_CAPACITY
        mandatory

      Auth_to_use

        Description:It specifies the unit that are
                    authorized to use given item.
        value class:UNIT
        mandatory
        multivalued
        inverse    :Nsn_No_Use
```

Figure 5.2 Identification Entity Class.

```
    Depot_of_Registry

      Description:Specifies the depot in which item
                  is registered.
      value class:DEPOT_STOCK_LEVEL
      mandatory
      multivalued
      inverse    :Nsn_No_Registered

    Supplier_Name

      Description:Supplier name that supplies the
                  item(s).
      value class:SUPPLIER_NAMES
      mandatory
      multivalued

    Past_Amount_Purchased

      Description:It specifies an amount that is
                  purchased in the past.
      value class:PAST_AMOUNT_PURCHASED
      match      :Amount of ORDER

    Sum_of_Used_UNITS

      value class:TOTAL_USED_IN_PAST
      mandatory

  identifier:

      Nsn_No + Document + Depot_of_registry
```

Figure 5.3 (cont'd.).

```
UNIT
  Description: All units in the Air Force that are use
               the item which are in the AF inventory.

  member attributes:

     Unit_Code

       value class: UNIT
       mandatory
       not changeable

     Superior_Comm

       description:The unit which has command and
                   control of this unit.
       value class:UNIT
       mandatory
       inverse    :Subordinate_Comm

     Nsn_NO_Use

       description:National stock number that are used
                   in the unit(s).
       value class:IDENTIFICATION
       inverse    :Auth_to_Use

     Qty_On_Hand

       value class:QUANTITY_ON_HAND

     Used_Amount

       description:Number of items that are previously
                   used in the unit.
       value class:TOTAL_USED_IN_PAST

     Req_Amount

       description:Specified number of items are
                   required in the unit for operational
                   readiness.
       value class:REQUIRED_AMOUNT_IN_UNIT

     Location

       description:Location of unit in geographical
                   coordinate system.
       value class:LOCATIONS

     Subordinate_Comm

       value class:UNIT
       inverse    :Superior_Comm

   identifier:

       Unit_Code + Nsn-No_Use
```

Figure 5.4 **Unit Entity Class.**

```
ORDER
   Description:Dependent up on the requests from the
               unit and depot, all ordered items by
               Department of Logistics of AF.

   member attributes

      Supp_Name

        description:Supplier name(s) that supplies the
                    item(s).
        value class:SUPPLIER_NAMES
        mandatory
        not changeable

      Nsn_No

        description:National stock number of item that
                    is ordered to supplier(s).
        value class:NATIONAL_STOCK_NUMBER
        mandatory

      Date

        description:Date of order
        value class:DATES
        mandatory

      Amount

        description:ordered amount for a given item.
        value class:ORDERED_AMOUNT

      Shipment_type

        value class:SHIPMENT
        multivalued

   identifier:

       Nsn_No
```

Figure 5.5 Order Entity Class.

```
DEPOT_STOCK_LEVEL
   description:Provides information about stock level
               of a given item in the depot.
   member attributes
      Depo_ID
        value class:DEPOT_ID
        mandatory
        not changeable
      Nsn_No_Register
        description:Different groups of items are
                    registered into different depots
                    such as communication items and
                    weapon items are stored into
                    different depots. This attribute
                    specifies registered item into
                    depot.
        value class:IDENTIFICATION
        mandatory
        inverse    :Depot_cf_Registry
      Stock_Amount
        description:Number of items that are currently
                    available as stock in the depot.
        value class:STOCK_STATUS
      Supplier_name
        value class:SUPPLIER_NAME
      identifier:
        Ns_No_Register + Depo_ID
```

Figure 5.6 Order Entity Class.

```
SUPPLIER
   description:All suppliers that are currently supply
               items to the AF
   member attributes:
      Supp_Name
        value class:SUPPLIER_NAME
      Ccuntry
        description:Country of supplier(s) that is/are
                    currently supply(ies) item(s).
        value class:COUNTRY
        mandatory
        multivalued
      City
        description:Supplier location as city.
        value class:CITIES
        multivalued
      Address
        description:Address of supplier that supplies
                    part.
        value class:ADDRESSES
   identifier
      Supp_Name
```

Figure 5.7 Supplier Entity Class.

NATIONAL_STOCK_NUMBER
interclass connection:subclass of STRING where it has 13 numbers which are divided into four(4) groups: 3020-00-001-0072

DOCUMENTATION
interclass connection:subclass of STRING where specified format.

QUANTITY_ON_HAND
interclass connection:subclass of STRING where format is positive integers.

TOTAL_USED_IN_PAST
interclass connection:subclass of STRING where format is positive integers.

MAX_AUTHORIZED_CAPACITY
interclass connection:subclass of STRING where format positive integers.

AUTHORIZED_TO_USE
interclass connection:subclass of STRING where format is five(5) characters

DEPO_OF_REGISTRY
interclass connection:subclass of STRING where format is five(5) characters

SUPPLIER_NAME
interclass connection:subclass of STRING where format is two(2) characters

PAST_AMOUNT_PURCHASED
interclass connection:subclass of STRING where format is positive integers.

UNIT
interclass connection:subclass of STRING where specified.

USED_AMOUNT_IN_UNIT
interclass connection:subclass of STRING where format is positive integers.

REQUEST_AMOUNT_IN_UNIT
interclass connection:subclass of STRING where format is positive integers.

LOCATION_OF_UNIT
interclass connection:subclass of STRING where specified.

Figure 5.8 **Domain of Attributes.**

```
DATES
   interclass connection:subclass of STRING where format
      is :
      month : number where =>1 and <=12
      "/"
      day   : number where integer and =>1 and <=31
      "/"
      year  : number where integer and =>1900 and <=2000
      where ( if ( month=4 or =5 or =9 or =11 ) then
      day <=30 ) and if ( month=2 then day<=29 )
      ordering by year,month,day.

ORDER_AMOUNT
   interclass connection:subclass of STRING where format
      is positive integers.

SHIPMENT
   interclass connection:subclass of STRING where speci-
      fied.

DEPOT_ID
   interclass connection:subclass of STRING where speci-
      fied.

STOCK_STATUS
   interclass connection:subclass of STRING where speci-
      fied.

COUNTRY
   interclass connection:subclass of STRING where speci-
      fied.

ADDRESSES
   interclass connection:subclass of STRING where speci-
      fied.

CITY
   interclass connection:subclass of STRING where speci-
      fied.
```

Figure 5.9 (cont'd).

VI. RELATIONAL MODEL

The relational mcdel was introduced to the database community by E.F. Codd (1970). This innovation stressed the independence of the relational representation from physical computer implementation such as ordering on physical devices, indexing, and using physical access paths. The model thus formalized the separation of the user view of data frcm its eventual implementation; it was the first model to do so. In addition, Codd proposed criteria for logically structuring relational databases and implementation-independent languages to operate on those databases. There have been many further developments in its theory and application. Relational design procedures have also received considerable attention in the last few years. P.A. Bernstein (1976) had proposed synthesizing relations from functional dependencies, and Fagin's work in 1977 then drew attention to the decomposition approach to design.

A. BASIC STRUCTURE OF THE RELATIONAL MODEL

Usefulness of the relational model in data analysis can be measured by considering several objectives. To meet the first objective-identify user requirements- the model must serve as a communication medium between the users and the computer personnel, giving them an interface that can be clearly and unambigucusly understood. The independence of this interface from computer implementation is of the utmost importance. The relational model uses tables to provide this interface. The tabular representation of relations satisfies the first objective of data analysis. The second objective, the conversion to physical implementation, is also satisfied

by the relational model. One obvious approach is to directly implement the relaticnal model on a machine. To do this a DBMS that supports the relational model must be available on the ccmputer system. A particular set of relations can then be directly declared by using the definitional language provided by the system. Direct conversion was not feasible when the relational model was first proposed by Codd in 1971, but today direct conversion from a relational specification to physical implementation is becoming increasingly possible. The third objective deals with the following criteria fcr logical data structures:

1. Each fact should be stored once in the database
2. The database should be consistent following database operation
3. The database should be resilient to change.

The first criterion not only removes storage redundancy but also improves database consistency. If the same fact is stored twice, it is possible that during execution of a complex operation, cnly one of the copy will be updated. The database then becomes inconsistent. In an inconsistent database, it is possible to get different database outputs for the same fact, thus creating a reliability problem. The second criterion requires that the database be consistent at all times. The third criterion deals with a different aspect. It is a consequence of the environment in which the database exists. This environment is usually in a state of constant change; ccnsequently, the database must be continually redesigned to meet continually changing user requirements.

1. Terminology

Informally, a database is made up of any number of relations. A relation is simply a two-dimensional table that

has several properties. First, the entries in the tables are single-valued; neither repeating groups nor arrays are allowed. Relations are flat files. Second, the entries in any cclumn are all of the same kind. Columns of a relation are referred to as attributes. Finally, no two rows in the table are identical in all attribute values and the order of the rows is insignificant. Figure 6.1 shows an example of a relation.

IDENTIFICATION

NIIN	FICHE-NO	FRAME-NO	ITEM-NO	
				-->Attribute
2335-00-679-0033	001	L10	05	-->Tuple
2835-00-682-5360	001	A10	05	-->Tuple
2345-00-680-9876	002	B77	08	-->Tuple

Figure 6.1 A Sample Relation Form.

Each row of the relation is known as a tuple. If the relation has n columns, then each row is referred to as an n-tuple. Also, a relation that has n columns or n attributes is said to be of degree n. Each attribute has a domain, which is a set of values that the attribute can have. Fcr example, in figure 6.1 the domain of the item-no is all positive integers less than 100. Sometimes it is possible that the domains of two attributes can be the same. To differentiate between attributes that have the same domain, each is a given a unique attribute name. The generalized format:

RELATION NAME (attribute name, attribute name,....)

IDENTIFICATION (NIIN, FICHE-NO, FRAME-NO, ITEM-NO), is called the relation structure. If we add constraints on allowable data values to the relation structure, we then have a relational schema [ref. 6].

a. Keys of Relations

The key is the attribute or set of attributes that uniquely identify tuples in a relation. A relation key is formally defined as a set of one or more relation attributes concatenated so that the following three properties hold for all time and for any instance of the relation:

1. Uniqueness: The set of attributes takes on a unique value in the relation for each tuple.
2. Nonredundancy: If an attribute is removed from the set of attributes,the remaining attributes do not possess the unique property.
3. Validity: No attribute in the key may be null.

It is possible for relations to have more than one relation key; each key is made up of a different set of attributes. The relation key is often called the candidate key. If a candidate key is the only key of the relation, it is generally referred to as primary key. When an attribute in one relation is a key of another relation, the attribute is called a foreign key. Foreign keys are important when defining constraints across relations. A prime attribute is an attribute that is part of at least one candidate key. A nonprime attribute is not part of any candidate key.

2. Consistency

The goal of relational design is to choose the relations that preserve consistency following database

operations and that store each fact at most once in the database. Relations that do this are said to be in normal form. In nonnormal relations, anomalies can arise after database tuple operation. The three tuple database operations are as follows:

1. ADD TUPLE (relation name, <attribute name>). This operation adds a new tuple to a relation. The attribute values of the tuple are given as part of the operation. For example:

add tuple (identification,<2835-00-678-4520,001,B01,05>) would add a new row to the relation in Figure 6.1. An add-tuple operation will not be allowed if it duplicates a relation key.

2. DELETE TUPLE (relation name,<attribute value>). This operation deletes a tuple from a relation. For example: delete tuple (IDENTIFICATION,<2335-00-679-0033,001,L10,05>) would delete the first row from the IDENTIFICATION relation.

3. UPDATE TUPLE (relation name,<old attribute values>,<new attribute values>). This operation changes the tuple in the relation. For example:

update tuple (IDENTIFICATION,<2835-00-682-5360,001,A01,05> , <2835-00-682-5360,002,L11,06>) This would change FICHE-NO, FRAME-NO, and ITEM-NO for NIIN value equal 2835-00-682-5360. Any update will not be allowed if it duplicates a relation key.

In a normal relational structure no anomalies arise after the application of any one of the three preceding operations with any set of attributes values.

3. Functional Dependency

Functional dependency [FD] is term derived from mathematical theory; it concerns the dependency of values of

one attribute or set of attributes on those of another attribute or set of attributes. Formally, a set of attributes X is functionally dependent on a set of attributes Y if a given set of values for each attribute in Y determines a unique value for the set of attributes in X. The notation Y-->X is often used to denote that X is functionally dependent on Y. Sometimes Y is called a determinant of the FD Y-->X. In the simplest case, both X and Y are made up of one attribute as shown in Figure 6.2.

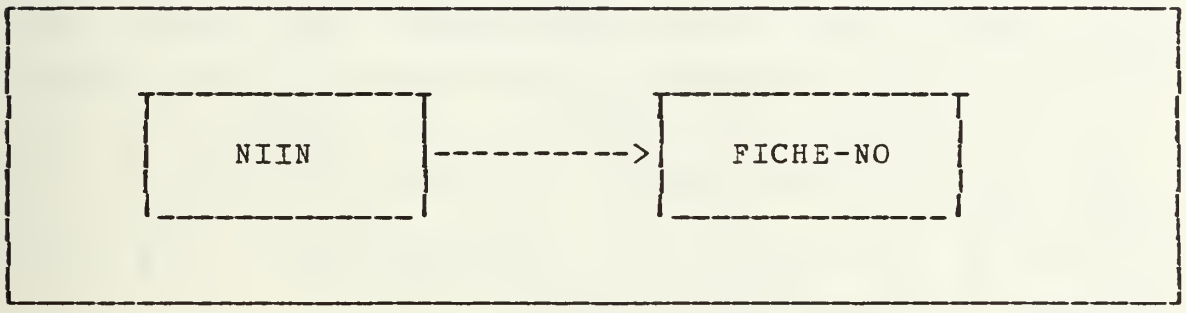

Figure 6.2 Functional Dependency Diagram.

It is also possible to have two attributes that are functionally dependent on each other. It is important to realize that functional dependency is a property of the information that is represented by relations. That is, functional dependency is not determined by the use of attributes in the relations or by the current contents of a relation.

Given a functional dependency Y-->X (where X and Y are both sets of attributes), a unique value for each attribute in X is determined only when the values for Y attributes are known. However, it is possible that values of X can be uniquely determined by only a subset of the attributes of Y. The term full functional dependency is used to indicate the minimum set of attributes in a determinant

of an FD. Formally a set of attributes X are fully functionally dependent on a set of attributes Y if

1. X is functionally dependent on Y.
2. X is not functionally dependent on any subset of Y.

Like functional dependency, full functional dependency is a property of the information that is represented by the relation.

4. Normal Forms

When determining whether a particular relation is in normal form, we should examine the FDs between the attributes in the relation. In the notation first proposed by C. Beeri and co-workers (1978), the relation is defined as made up of two components: the attributes and the FDs between them. R1 = ({X,Y,Z}, { X-->Y , X-->Z }) The first component of the relations is the attributes, and second component is the FDs. For example,

IDENTIFICATION = ({NIIN,FICHE-NO,FRAME-NO,ITEM-NO} , {NIIN-->FICHE-NO , NIIN-->FRAME-NO , NIIN-->ITEM-NO})

The functional dependencies between attributes in a relation are obviously important when determining the relation's key.

There are a number of normal forms as shown in Figure 6.3. Relations are in first normal form (1NF) if all domains are simple. In other words all legitimate relations are in 1NF.

A relation is normalized by replacing the nonsimple domains with simple domains. A relation R is in second normal form (2NF) if every nonprime attribute of R is fully functionally dependent on each relation key.

A relation R is in third normal form if it has the following properties:

1. The relation R is in second normal form, and

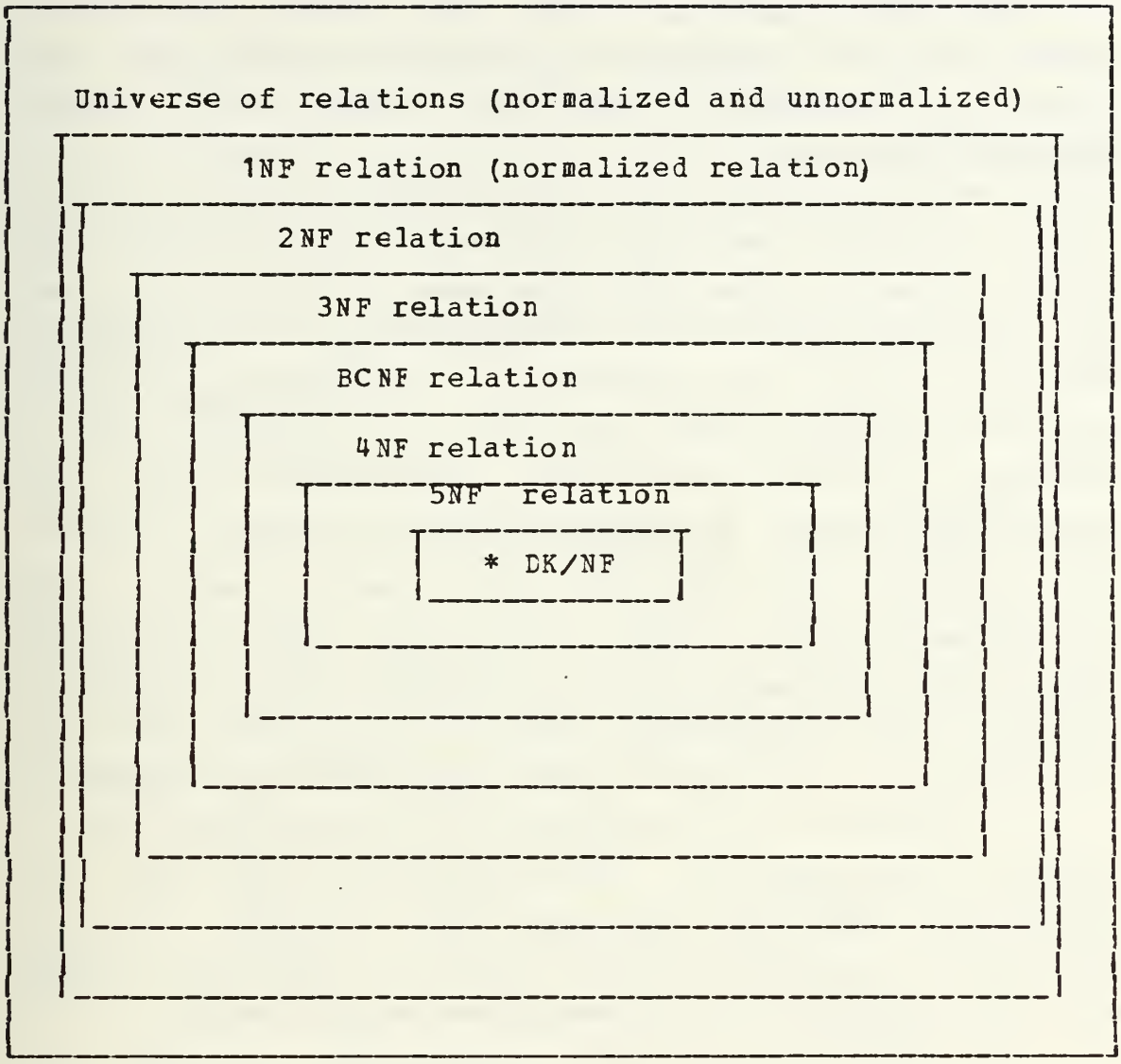

Figure 6.3 Normal Forms.

2. The nonprime attributes are mutually independent; that is, it has no transitive dependency.

In other words, a relation R is in third normal form (3NF) if and only if it is in 2NF and every nonprime attribute is nontransitively dependent on the primary key. For example, suppose

R = ({A,B,C,D}, {AB-->C, C-->D}) AB is primary key. AB-->C C-->D by transitivity AB-->D hence relation R is not in 3NF, because, there is a transitivity between nonprime attributes.

In the definition of the third normal form we assumed that the relation had only one relation key. Problems arise with the definition when applied to relations that have more than one relation key. The original definition of 3NF was modified by a stronger definition which was proposed by Boyce and Codd. It is known as BCNF. A relation R is in BCNF (Boyce/Codd Normal Form) if and only if every determinant is candidate key. For example, suppose

R = ({A,B,D,E} , {A-->BED , D-->A}) Here relation R will be in BCNF if both A and D are keys of R. Formally, multivalued dependency is defined as follows; in relation R(X,Y,Z), X === > Y if each X value is associated with a set of Y values in a way that does not dependent on Z values.

A relation is in 4NF if it is in BCNF and has no multivalued dependencies. This definition means that if a relation has multivalued dependency and is in 4NF, then the multivalued dependencies have a single value. In others words, all independent attributes have single value.

A relation is in 5NF if and only if every join dependency in a relation R is implied by the candidate keys of relation R.

A relation is in Domain-Key normal form (DK/NF) if every constraint on the relation is a logical consequence of the definition of the keys and domains. A constraint is any rule on the static value of the attributes that is precise enough that we can evaluate whether or not it is true. Examples of the constraints are inter-relation constraints, functional dependencies, multivalued dependencies, and join dependencies. DK/NF means that if we can define keys and

domains in such a way that all constraints will be satisfied, then modification anomalies are impossible. Unfortunately, there is no known way to convert a relation to DK/NF automatically, nor it is even known which relations can be converted to DK/NF. In spite of this, DK/NF can be exceedingly useful for practical database design.

B. ADVANTAGES AND DISADVANTAGES OF RELATIONAL MODELS

1. Advantages

a. Simplicity

The end user is presented with a simple data model. User requests are formulated in terms of the information content and do not reflect any complexities due to system-oriented aspects. A relational data model is what the user sees, it is not necessarily what will be implemented physically[Ref.11].

b. Nonprocedural Request

Because there is no positional dependency between the relations, requests do not have to reflect any preferred structure and therefore can be nonprocedural.

c. Data Independence

This should be one of the major objectives of any database management system. The relational data model removes the details of the storage structure and access strategy from the user interface. The relational model provides a relatively higher degree of data independence than do network and hierarchical models. However, the design of the relations must be complete and accurate for making use of this property of the relational model.

d. Theoretical Foundation

The relational data model is based on the well-developed mathematical theory of relations. The rigorous method of designing a database using normalization gives this model a solid foundation. This kind of foundation does not exist for the other two models.

2. Disadvantages

A disadvantage sometimes cited for a relational model is machine performance. With present-day hardware the JOIN operation is likely to take substantial machine time. It is feasible with small relations, but some commercial files are hundreds of millions of bytes long. In understanding the performance issue, it is very important to remember that the relations and the operations on them such as the JOIN will never occur physically. Instead, equivalent results will be produced by means of pointer structures or indices. It appears today that technological improvements in providing faster and more reliable hardware may solve this problem.

VII. RELATIONAL DATABASE DESIGN

The relational model is attractive in the database design because it provides formal criteria for logical structure, namely, normal form relations. The problem, then, is to choose a design procedure to produce normal form relations. Two different approaches have been proposed:

1. Decomposition procedures: These commence with a set of one or more relations and decompose nonnormal relations in this set into normal forms.
2. Synthesis procedures: These commence with a set of functional dependencies and use them to construct normal form relations.

Most designs commence with an information gathering phase in which a set of data elements and FDs between them are identified. The information is then used to produce normal relations. On the other hand, one could conceive of a procedure where all the data attributes are considered to form one relation, which is then decomposed in subsequent design steps.

A. RELATIONAL DESIGN CRITERIA

Beeri and co-workers (1978) have identified three relational design criteria:

1. SEPARATION: The original specifications are separated into relations that satisfy certain conditions.
2. REPRESENTATION: The final structure must correctly represent the original specifications.
3. REDUNDANCY: The final structure must not contain any redundant information.

The separation criteria is that the database must be separated into a number of normal relations. The other two criteria are relatively general. In specific terms each can be applied to attributes, FDs, or data. Here, criteria will be defined more specifically. For example, given the relation R = ({A,B,C} , {A --> B , A --> C}).

Here R comprises three attributes,A,B, and C. The functional dependency between these attributes are A-->B and A-->C. The notation used to describe the input and output of the design process is Sin and Sout. Sin and Sout are sets of relations. Here Sin is the input to the design process and Sout is the output of the design process.

1. Representation Criteria

One goal of any design process is to produce an output design, Sout, to accurately represent Sin. All the relations in Sout must satisfy the conditions for normal form. Beeri and co-workers (1978) have defined three representation criteria for the representation of Sin by Sout:

1. REP1: The relations Sout contain the same attributes as Sin.
2. REP2: The relations Sout contain the same attributes and the same FDs as Sin.
3. REP3: The relations in Sout contain the same attributes and the same data as Sin.

REP1 requires all the attributes in Sin to also appear in the relations in Sout. But it does not consider any dependencies between the attributes. According to REP2 Sin will contain a set of attributes and a set of functional dependencies. Sout will also contain a set of attributes and a set of FDs. Representation REP2 requires that each FD in Sin be either:

1. Contained as an FD in one of the relations in Sout or

2. Derived from the FDs in the relations in Sout, using the FD inference rules. For example in Figure 7.1, Sin = ({A,B,C} , { A --> B , C --> B }) and Sout = (R2,R3) where R2=({A,B},{A-->B}) and R3=({B,C},{C-->B}).

Thus R2 and R3 constitute the decomposition by projection of Sin. Each of the functional dependencies in Sin is contained in Sout; hence we can say that Sout is a REP2 representation of Sin. It is interesting that Figure 7.2 shows a decomposition that is not a REP2 representation of Sin[Ref.10].

Figure 7.1 includes a relation R1 that is decomposed by projection into two relations, R2 and R3, in Sout. Note that R2 and R3 do not contain the same information as Sin since different responses are obtained to the same question applied to Sin and Sout. Hence Sout is not an REP3 representation of Sin. Because if we ask: To what c is a1 related? In Sin the answer is {c1}; in Sout the answer is {c1,c2}. This join in Figure 7.1, contains additional tuples to those of Sin and is sometimes known as a CROSS JOIN. Note that in Figure 7.2 the two relations Y1 and Y2 in Sin are an REP3 representation of Sin because their join contains exactly the same tuple as in the original relation, R.

2. Lossless Decomposition

Formally, a lossless decomposition can be described as follows. The decomposition of a relation R(X,Y,Z) into relations R1 and R2 is defined by two projections: R1 = projection of R over X,Y and R2 = projection of R over X,Z where X is the set of common attributes in R1 and R2. The decomposition is lossless if R = join of R1 and R2 over X. The decomposition is lossly if R is a subset of the Join of R1 and R2 over X.

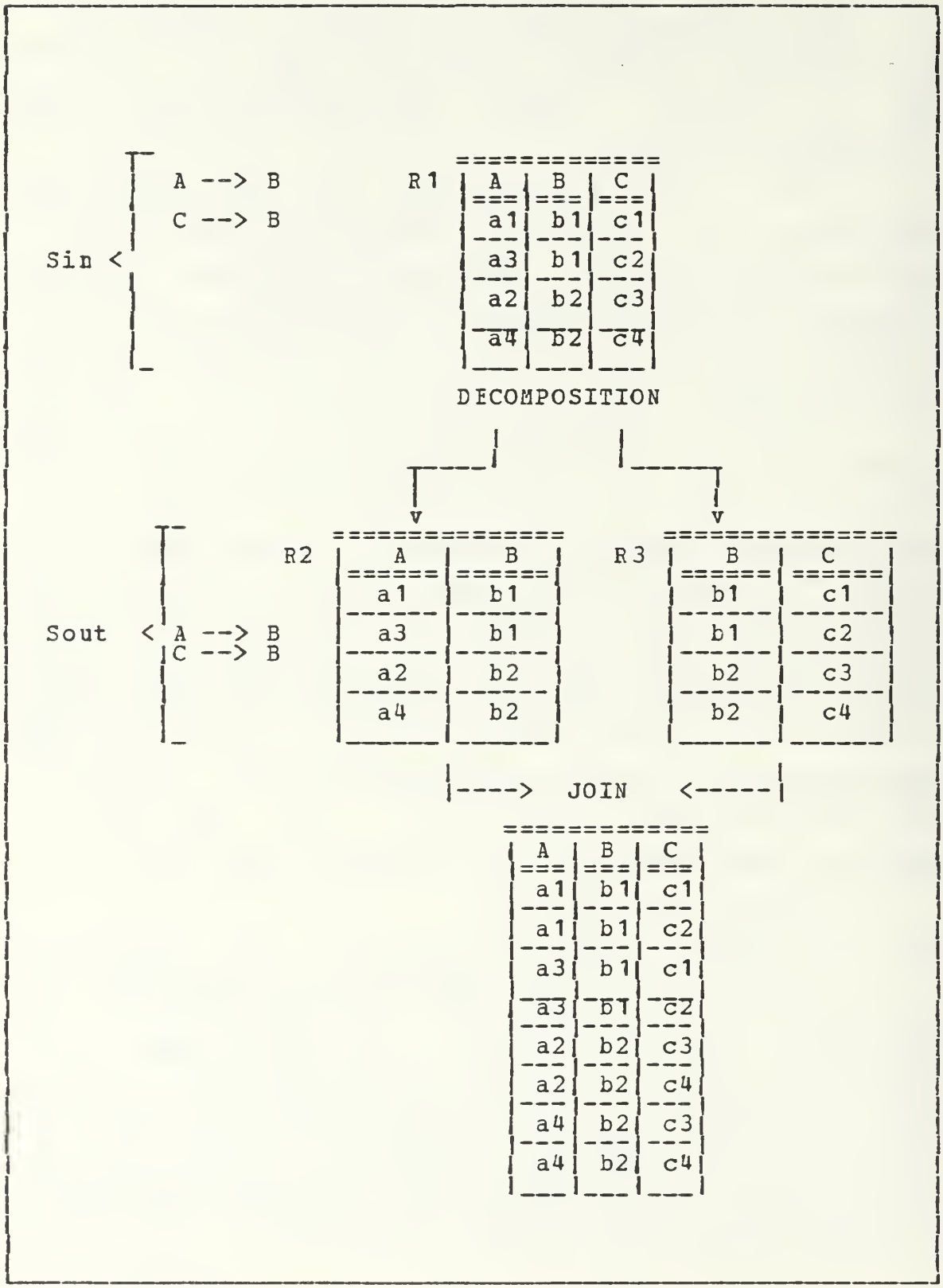

Figure 7.1 Decomposition.

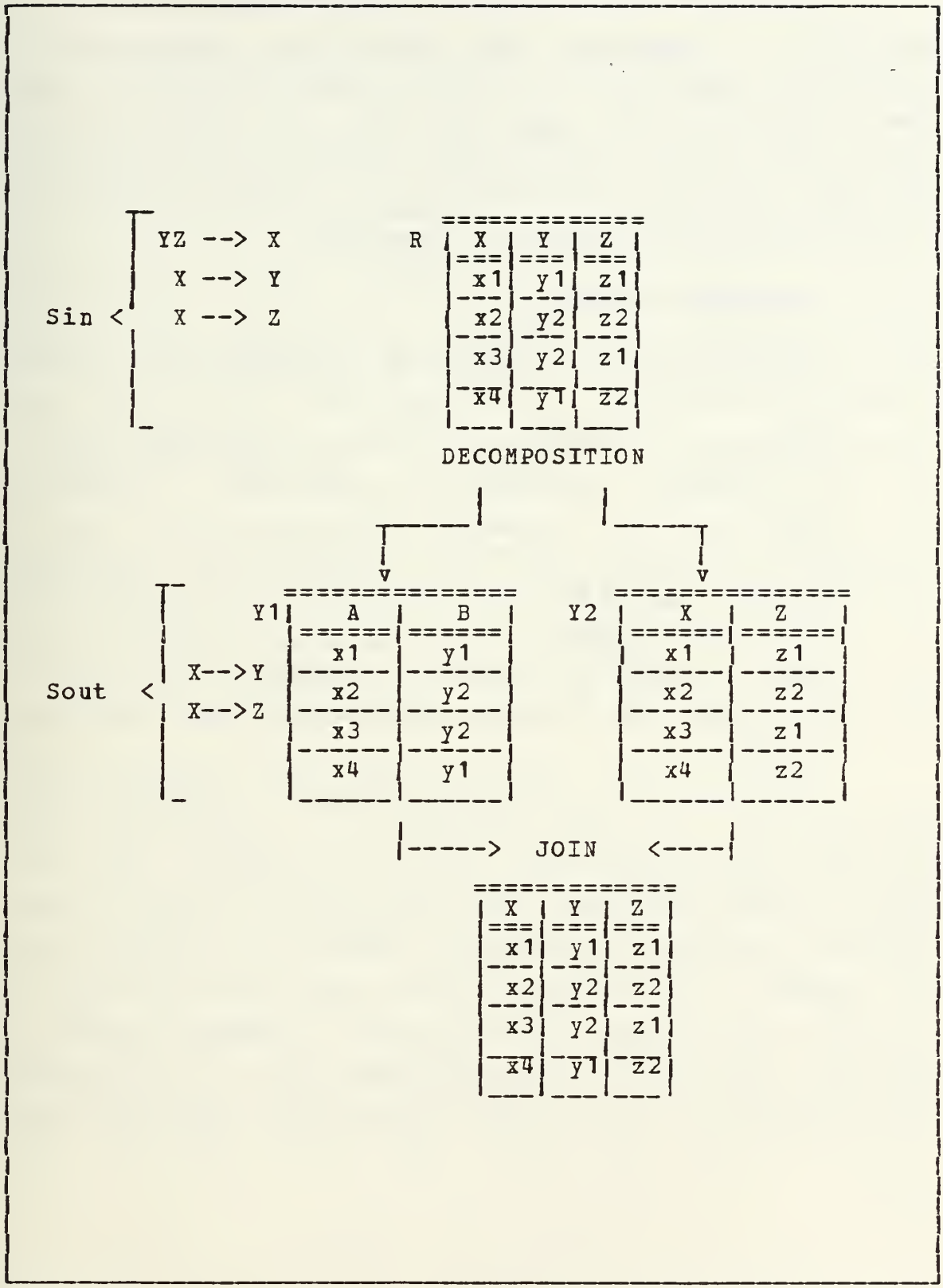

Figure 7.2 Decomposition.

CCNDITIONS: The decomposition of R(X,Y,Z) in R1(X,Y) and R2(X,Z) is lossless if for attribute X, common to both R1 and R2, either X-->Y or X-->Z. Thus in figure 7.1 the common attribute of R2 and R3 is B, but either B-->A or B-->C is true, hence decomposition is lossly. In Figure 7.2 the ccmmon attribute of Y1 and Y2 is X, both X-->Y and X-->Z is true, hence decomposition is lossless.

3. Redundancy Criteria

Redundancy can be defined in various ways. One set of redundancy criteria is as follows [Ref.7]:

1. RED1 : A relation in Sout is redundant if its attributes are contained in the other relations in Scut.
2. RED2 : A relation in Sout is redundant if its FDs are the same or can be derived from the FDs in the cther relations in Scut.
3. RED3 : A relation in Sout is redundant if its content can be derived from the contents of other relaticns in Sout.

RED1 is not a powerful criterion, because during separaticn it is cften necessary to create separate relations that represent FDs between attributes, which may appear in other relations. RED2 and RED3 can be quite useful criteria. Any design algorithms should in particular avoid RED3 because it would keep the same data in more than one relation. Such relations could all be in normal form and no anomalies would occur in relations. But, interrelation anomalies would arise if the same fact were updated in one relation but not the other. RED2 would cause the same problem.

B. RELATIONAL DESIGN PROCEDURE

It is interesting to note that in Figure 7.1 the design Sout is on REP2 but not on REP3 representation of Sin whereas in Figure 7.2 the design Sout is on REP3 but not on REP2 representation of Sin. This situation creates problems of relational research; namely, to find a design procedure that yields an Sout that is both on REP2 and REP3 representation of Sin. Similarly, design procedures should aim to reduce redundancy, but here again different design procedures can result in either RED2 or RED3 representations of Sin [Ref.8].

There are two classes of algorithms: decomposition and synthesis. Decomposition algorithms commence with one relation and successively decompose it into normal form relations. The concepts of 3NF and BCNF are not sufficient for decomposition algorithms, so the ideas of multivalued dependency and a 4NF have to be introduced.

Synthesis algorithms use FDs to produce normal form relations. For these algorithms to be successful it is necessary to ensure that:

1. FDs in Sin correctly represent user semantics,
2. Algorithms can be devised to produce relations in Sout that correctly and nonredundantly represent Sin.

If synthesis algorithms are to be effective, their input must describe those nonfunctional relationships that cannot be expressed as FDs between attributes. Perhaps the best-known synthesis algorithm is the one devised by Bernstein. It is premised on grouping all FDs with the same determinant and constructing a relation for each such group.

C. PHYSICAL DESIGN OF INVENTORY DATABASE

1. Mapping from SDM into Relational Model

The logical design of the inventory database cannot be used as the physical design of a relational database. For example, in the IDENTIFICATION and UNIT records, there are some multivalued attributes which are not allowed in a relation. The relations must be transformed so that each attribute has only one value per tuple. Also, the logical design in Chapter 5 allows tuples to be contained in other tuples which cannot be done physically. Relations in the logical design have to be redefined to eliminate this problem.

Consider the relations UNIT and IDENTIFICATION. Actually, Auth_to_use of IDENTIFICATION is a collection of tuples representing UNIT which are using a specified item. We can eliminate Auth_to_use of IDENTIFICATION, because, whenever we need this information we can get it by use of the Data Manipulation Language (DML). It is possible to construct contained tuples by DML joins. In this case, Auth_to_use will be constructed and not stored.

The process just described can be used to transform the logical schema into a relational schema. All contained tuples have been replaced using the same logic. Auth_to_use of IDENTIFICATION is deleted and interrelation constraints are added. Similarly, Depot_of_registry of IDENTIFICATION and Subordinate_comm of UNIT and Past_amount_purchased of IDENTIFICATION are deleted and interrelation constraints are added.

The resulting design is shown in Figure 7.3 and 7.4. Figure 7.3 shows relation, attributes, and interrelation constraints, and Figure 7.4 shows the domains and attribute-domain correspondences.

```
PART-IDENTIFICATION(Nsn_No,Tot_Qty-On_Hand,
                    Sum_of_used_unit,Max_Auth_Qty_Hand)
                    KEY : Nsn_no

DOCUMENT_IDENTIFICATION(Nsn_No,Document,Supp_name)
                    KEY : Nsn_No

UNIT_INVENTORY(Unit_Code,Nsn_No_Use,Qty_On_Hand,
               Used_amount,Req_amount)
               KEY : Unit_Code + Nsn_no_Use

UNIT_ID(Unit_Code,Superior_Comm,Location)
       KEY : Unit_Code

ORDER(Nsn_no,Supp_name,Date,amount,Ship_Type)
     KEY : Nsn_no

DEPOT_STOCK_LEVEL(Depo_Id,Stock_amount,Supp_name,
                  Nsn_No_Registry)
                  KEY : Depo_ID

SUPPLIER(Supp_name,Country,Address,City)
        KEY : Supp_name
```

Figure 7.3 Records of Relational Schema.

Attribute	Domain
Nsn_No	NATIONAL STOCK NUMBER
Dccument	DOCUMENTATION
Tot_qty_on_hand	TOTAL QUANTITY
Max_auth_qty_hand	MAXIMUM QUANTITY
Sum_of_used_units	SUM OF USED
Supp_name	S_NAME
Unit_code	UNIT_NAME
Superior_comm	UNIT_NAME
Nsn_no_use	NATIONAL STOCK NUMBER
Qty_on_hand	TOTAL QUANTITY
Used_amount	SUM OF USED
Req_amount	REQUIRED AMOUNT
Location	LOCATIONS
Date	DATES
Amount	ORDER_AMOUNT
Ship_type	SHIPMENT_TYPE
Depo_id	D_NAME
Nsn_no_registry	NATIONAL STOCK NUMBER
Stock_amount	TOTAL AMOUNT
Ccuntry	C_NAME
Address	ADDRESSES
City	CITIES

Figure 7.4 Attributes and Domains.

VIII. SYSTEM R: RELATIONAL APPROACH TO DATABASE MANAGEMENT

System R is a database management system which provides a high level relational data interface. The system provides a high level of data independence by isolating the end-user as much as possible from underlying storage structures. The system permits definition of a varity of relational views on common underlying data. Data control features are provided, including authorization, integrity assertions, triggered transactions, a logging and recovery subsystem, and facilities for maintaining data consistency in a shared-update environment. System R supports a relational database,i.e., a database in which all data is perceived by users in the form of tables. All access to this database is via a data sublanguage called SEQUEL.

A. ARCHITECTURE AND SYSTEM STRUCTURE

Figure 8.1 gives a functional view of the system including its major components and interfaces. The Relational Storage Interface (RSI) is an interface which handles access to single tuples of base relations.

This interface and its supporting system, the Relational Storage System (RSS), is actually a complete storage system in that it manages devices, space allocation, storage buffer, transaction consistency and locking, deadlock detection, backout, transaction recovery, and system recovery. Also it maintains indices on selected fields of base relations and pointer chains across relations.

The Relational Data Interface (RDI) is the external interface which can be called directly from a programming language, or used to support various emulators and other

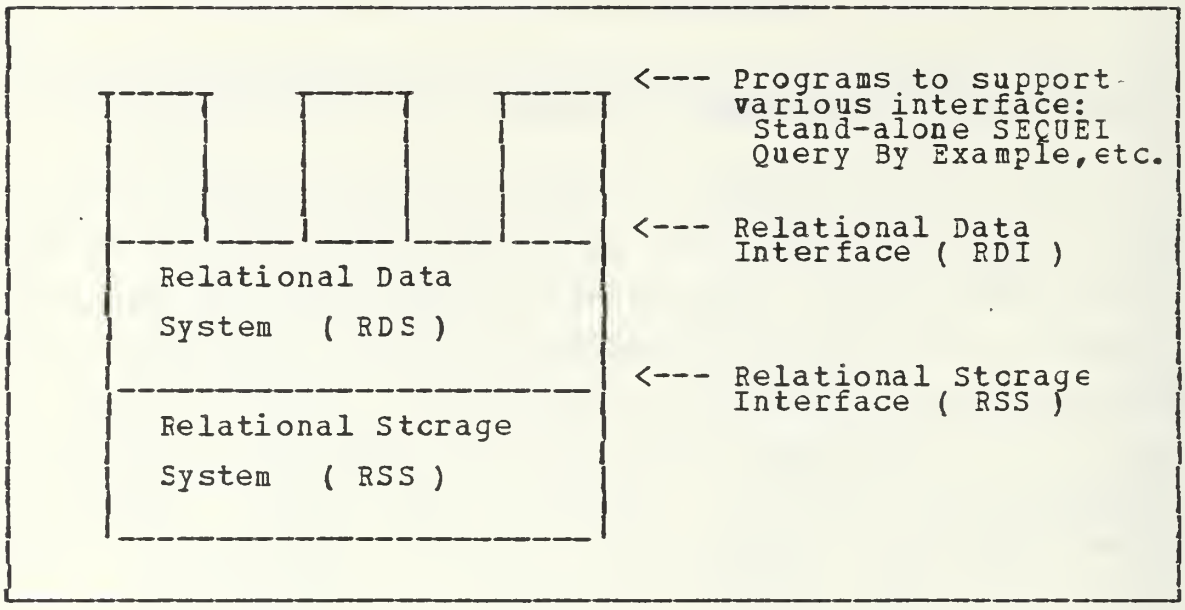

Figure 8.1 Architecture of System R.

interfaces. The Relational Data System (RDS), which supports the RDI, provides authorization, integrity enforcement, and support for alternative views of data. The high level SEQUEL language is embedded within the RDI and is used as the basis for all data definition and manipulation. In addition, the RDS maintains the catalogs of external names, since the RSS uses only system generated internal names. The RDS contains an optimizer which chooses an appropriate access path for any given request from among the paths supported by the RSS. RSS and RDS will be evaluated in detail the following next two sections.

B. THE RELATIONAL DATA SYSTEM

The Relational Data Interface (RDI) is the principal external interface of System R. The data definition facilities of the RDI allow a variety of alternative relational views to be defined on common underlying data. The RDS is the subsystem which implements the RDI. The RDI

consists of a set of operators which may be called from PL/I or other host programming languages. All facilities of the SEQUEL data sublanguage are available at the RDI by means of the RDI-called SEQUEL. SEQUEL is designed to be used both as a stand-alone language for interactive users and as a data sublanguage embedded in a host programming language such as PL/I. In the latter case the SEQUEL statements in the program are identified by a precompiler which replaces them with valid PL/I calls to a run-time module which provides the environment for executing an application program that has been through the precompilation process. The precompilation process is described below[Ref.6].

1. The precompiler scans the source program and locates the embedded SEQUEL statements.
2. For each statement it finds, the precompiler decides on a strategy for implementing that statement in terms of RSI operations. Having made its decisions, the precompiler generates machine language routines (including calls to the RSS) that will implement the chosen strategy. The set of all such routines together constitutes the access module for the given source program. The access module is itself stored in the database.
3. The precompiler replaces each of the original embedded SEQUEL statements by an ordinary PL/I statement to the run-time module of the RDS.

The modified source program can now be compiled by the PL/I compiler in the normal way. This process is depicted in Figure 8.2.

In terms of query facilities, SEQUEL provides extensive query facilities based on English key words. As a relational DBMS we have ORACLE in our school. In terms of Query facilities there is no big difference between System R and ORACLE. Query, data manipulation, and data definition

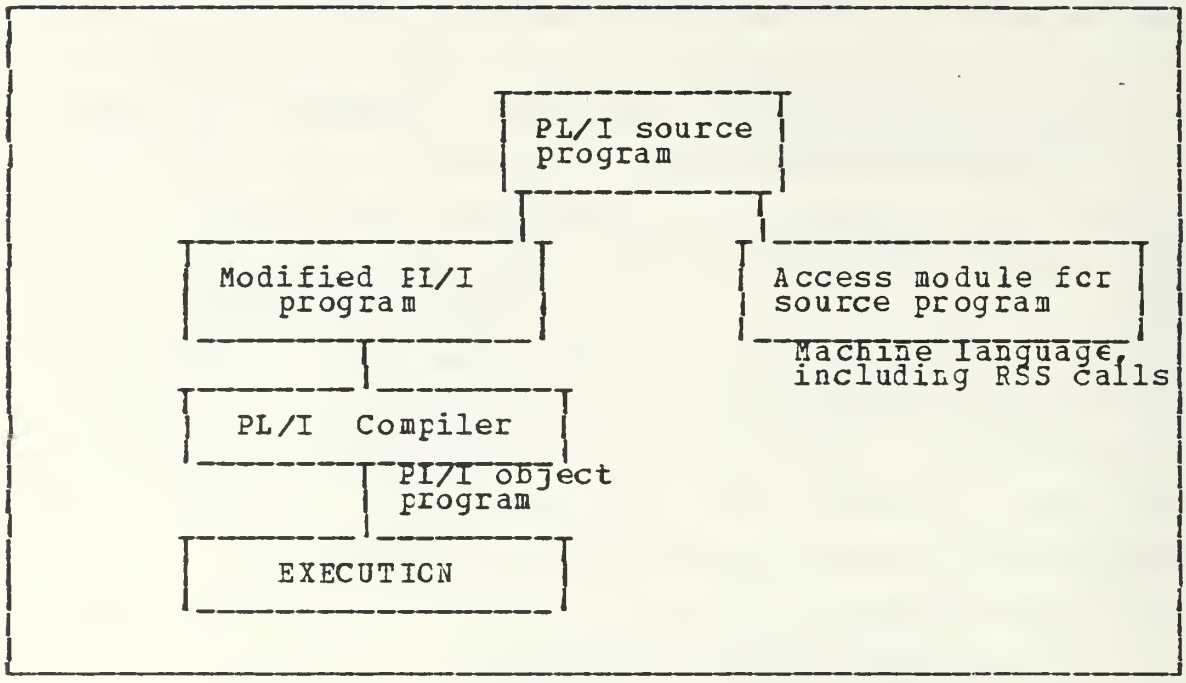

Figure 8.2 Precompilation Process.

facilities of ORACLE will be illustrated over the Inventory Database by a series of examples in Chapter 9.

1. Data Definition Facilities

The primary data structure in System R is the Base Relation (Base Table). The base relation is a table that has its own independent existence and is represented in the physical database by a stored file. Base table can be created at any time by executing the SEQUEL DDL statement CREATE TABLE, which takes the general form:

```
CREATE TABLE base-table-name
        (field-definition , ..... )
      { IN SEGMENT segment-name }
```

where a field-definition, in turn, takes the form

```
field-name ( data-type {, NONULL } )
```

Successful execution of the CREATE TABLE statement causes a new, empty base table to be created in the specific segment

with the specific base-table-name and specific field definition. The user may now proceed to enter data into that table using the SEQUEL INSERT statement. A System R database is partitioned into a set of disjoint SEGMENTS which provides a mechanism for controlling the allocation of storage and the sharing of the data among users. Any given base table is wholly contained within a single segment and indices on that base table are also contained in that same segment. However, a given segment may contain several base tables and their indices. A public segment contains shared data that can be simultaneously accessed by multiple users. A private segment contains data that can be used by only one user at a time. If the CREATE TABLE statement does not specify the segment, then the base table will go in a private segment belonging to the user that issued the CREATE TABLE. This specification is an option in the CREATE TABLE statement. Each field definition in CREATE TABLE includes three items: A field-name, a data-type for the field, and optionally a NONULL specification. The field name has to be unique within the base table. The System R supports the concept of nonull field values. Null is a special value that is used to represent "value unknown" or "value inapplicable".

By using the EXPAND TABLE statement, an existing base table can be expanded at any time by adding a new column at the right :

```
EXPAND TABLE base-table-name
        ADD FIELD filed-name ( data-type )
```

The important point is that the specification NONULL is not permitted in EXPAND TABLE. It is also possible to destroy an existing base table at any time:

```
DROP TABLE base-table-name
```

All records in the specific base table are deleted, all indexes and views on that table are destroyed, and the table

itself is then also destroyed; that is, its description is removed from the dictionary and its storage space is released[Ref. 7].

The query power of SEQUEL may be used to define a view as a relation derived from one or more other base tables. This view may then be used in the same ways as a base table: queries may be written against it, other views may be defined on it, and in certain circumstances described below, it may be updated. Any SEQUEL query may be used as a view definition by means of a DEFINE VIEW statement:

```
DEFINE VIEW view-name
        { ( field-name , .......) }
        AS SELECT - statement
```

Views are dynamic windows on the database as shown in Figure 8.3. In System R, a view that is to accept updates must be derived from a single base table. Moreover, it must satisfy the following constraints:

1. Each distinct row of the view must correspond to a distinct and uniquely identifiable row of the base table.
2. Each distinct column of the view must correspond to a distinct and uniquely identifiable column of the base table. If a view does satisfy constraints 1 and 2, then any update against it can easily be mapped into an update on the corresponding base table.

There is another SEQUEL command for data definition facility: KEEP TABLE. It causes a temporary table to become permanent. Normally, temporary tables are destroyed when the user who created them logs off.

2. Data Control Facilities

System R has extensive data control facilities that enable users to control access to their data by other users,

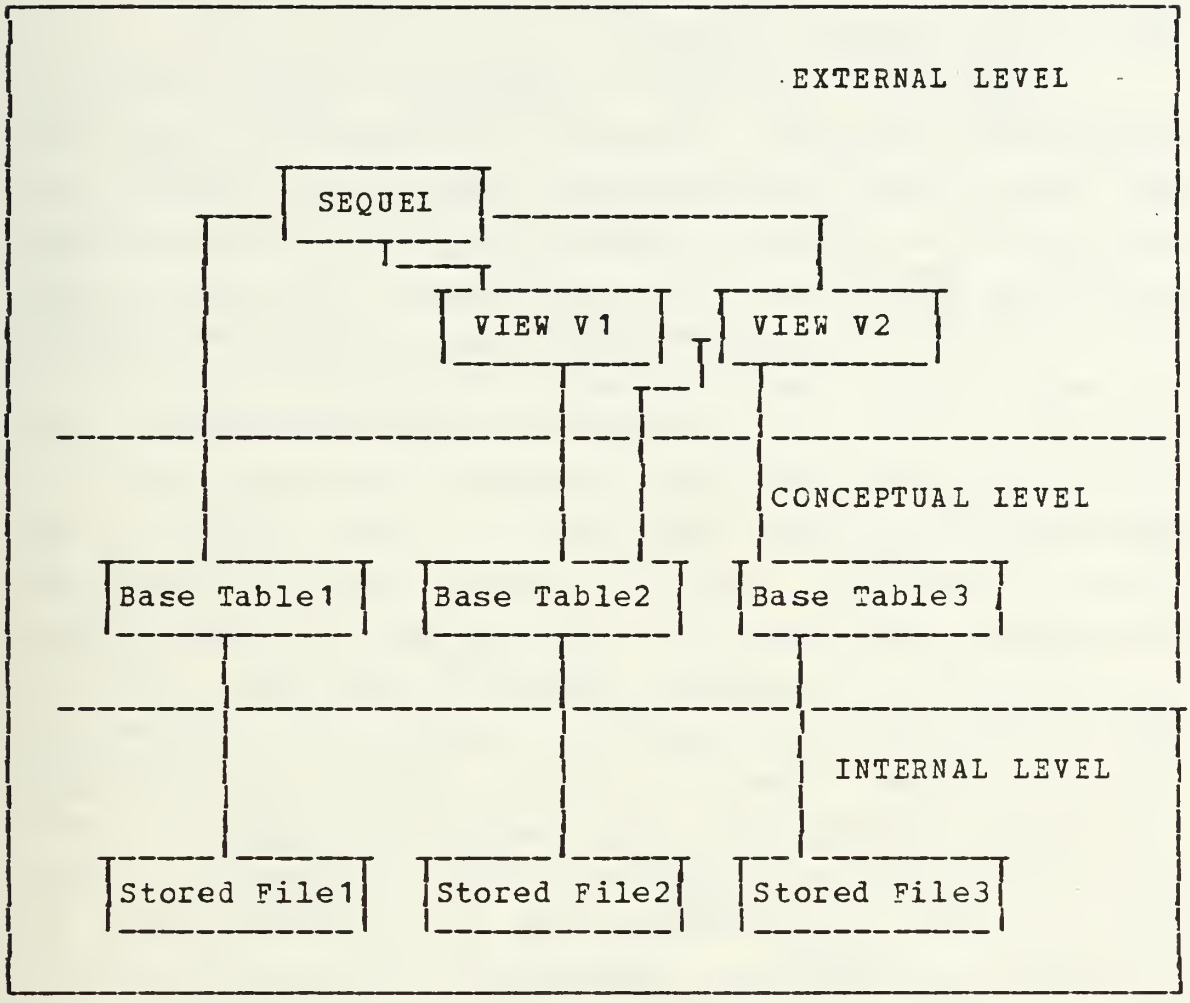

Figure 8.3 System R as Seen by an User.

and to exercise control over the integrity of data values. The data control facilities have four aspects: transactions, authorization, integrity assertions, and triggers.

A transaction is a series of the statements which the user wishes to be processed as an atomic act. The meaning of the "atomic" depends on the level of consistency specified by the user. The user controls transactions by the operator BEGIN-TRANS and END-TRANS. The user may specify save points within a transaction by the operator SAVE. As long as a transaction is active, the user may block up to

the begining of the transaction or to any internal space point by the operator RESTORE.

System R allows for an extremely simple method of authorization checking. System R maintains two tables for the use of the authorization subsystem: SYSAUTH and SYSCOIAUTH. The SYSAUTH table has up to two rows for each combination of relation (base or view) and user. The columns in the SYSAUTH table correspond to user ID, base relation or view name, type (base or view), a column for each of the privileges on the relation (Y OR N) and a column for grant option (Y or N). For each relation on which a user is authorized to perform some action, there are up to two tuples in SYSAUTH: one for grantable and the other for non-grantable privileges. In case the user has update rights on a relation, the table SYSCOLAUTH indicates precisely those columns of the relation on which the user has the update privilege. These two tables, SYSAUTH and SYSCOIAUHT, are updated whenever a new base relation or view is created or an authorized user executes a GRANT statement, thereby granting a set of privileges to one or more other users. The two tables are referenced immediately before the execution of any SEQUEL statement[Ref. 5].

The third important aspect of data control is that of integrity assertions. Any SEQUEL logical expression associated with a base table or view may be stated as an integrity assertion. At the time an assertion is made by an ASSERT statement, its truth is checked; if true, the assertion is enforced until it is explicitly dropped by a DROP ASSERTION statement. Any data modification by any user which viclates an active integrity assertion is rejected. Assertions may apply to individual tuples or to sets of tuples.

The fourth aspect of data control, triggers, is a generalization of the concept of assertion. A trigger causes

a prespecified sequence of SEQUEL statements to be executed whenever some triggering events occurs. The triggering event may be retrieval, insertion, deletion, or update of a particular base table or view. RDI can monitor such events by simply scanning a transaction for a SEQUEL statement that corresponds to a particular triggering event. After each of these statements, immediately a call statement is included to invoke the appropriate trigger routine.

3. Data Manipulation Statements

The RDI facilities for insertion, deletion, and update tuples are also provided via the SEQUEL data sublanguage. SEQUEL operates on both base tables and views. It can be used to manipulate either one tuple at time or a set of tuples with a single command. By using these facilities, it is possible to assign the result of a query to newly created relation.

An insertion statement in SEQUEL may provide only some of the values for the new tuple, specifying the names of the field which are provided. Fields which are not provided are set to the null value. The physical position of the new tuple in storage is influenced by the "clustering" specification made on associated RSS access paths.

Deletion is done by means of a DELETE statement accompanied by a WHERE clause. The WHERE clause specifies the conditions that must be satisfied by the records to be deleted. The RDI can translate the UPDATE statements in one of two ways:

1. By using the RETRIEVE command to determine the addresses of the selected records, and then using the REPLACE command to modify these records one at a time.
2. By using the REPLACE command to modify all the selected records simultaneously.

Which of these two methods is to be used depends on the actual SEQUEL statement. If the SET clause makes identical changes to all the selected tuples, then only the second method should be used. The SEQUEL assignment statement allows the result of a query to be copied into a new permanent or temporary relation in the database. This has the same effect as a query followed by the RDI operator KEEP. The execution of an assignment statement by the RDI is done in two parts:

1. The records satisfying the query are retrieved,
2. A new relation is created with the records retrieved in (1). These records are then stored in the database.

A series of examples will be given for inventory database by using ORACLE relational DBMS in Chapter 9.

4. Optimizer

The objective of the optimizer is to find a low cost means of executing a SEQUEL statement, given the data structures and access paths available. The optimizer attempts to minimize the expected number of pages to be fetched from the secondary storage into the RSS buffers during execution of the statement. Only page fetches made under the explicit control of the RSS are considered. If necessary, the RSS buffers will be pinned in real memory to avoid additional paging activity caused by the operating system such as the VM/370 operating system. The cost of the CPU instructions is also taken into account by means of an adjustable coefficient which is multiplied by the number of tuple comparison operations to convert to equivalent page accesses. The adjustable coefficient can be adjusted according to whether the system is computation-bound or I/O bound[Ref. 6].

After analyzing any SEQUEL statement, the optimizer produces an Optimized Package (OP) containing the parse tree and a plan for executing the statement. If the statement is a query, OP is used to materialize tuples as they are called for by the fetch command. If the statement is a view definition, the OP is stored in the form of a Pre-Optimized Package (POP) which can be fetched and utilized whenever an access is made via the specified view. If any change is made to the structure of the base table or to the access paths maintained on it, the POPs of all views defined on that base table are invalidated, and each view must be reoptimized from its defining SEQUEL code to form a new POP.

C. THE RELATIONAL STORAGE SYSTEM

The RSS is essentially a powerful access method. Its primary function is to handle all details of the physical level and to present its user with an interface called the RSI. The user of the RSS is normally not a direct user, but is code generated by the RDS in compiling some SEQUEL statement. The RSI was specifically designed to be a good target for the SEQUEL compiler.

As shown in Figure 8.3, the basic data object at the RSI is the stored file which is the internal representation of a base table. Rows of the table are represented by records of the file; the stored records within one stored file need not be physically adjacent in storage. An arbitrary number of indexes over any given stored file is supported by the RSS, thus providing the additional access paths to that file. The RSS objects (stored files,indexes,etc.) and the associated operators together constitute the Research Storage Interface(RSI). As mentioned above it is the interface used as the target by the RDS in precompiling SEQUEL requests. The user of the RSI needs to know what stored files and

indexes exist, and must specify the access path(index or system sequence) to be used in any given RSI access request.

1. Segments

In the RSS, all data is stored in a collection of the logical address space called SEGMENTS, which are employed to control physical clustering. Segments are used for storing user data, access path structures, internal catalog information, and intermediate results generated by the RDS. All the tuples of any relation must reside within a single segment chosen by the RDS, but a given segment may contain several relations. Three types of segment are supported, each with its own combination of functions and overhead: shared (or public) , private, and temporary data segments. Basically data in shared segments are recoverable and sharable; data in private segments are recoverable but not sharable; and data in temporary segments is neither recoverable nor sharable. Segment type is fixed at the time of the system installation and cannot be changed. Each segment consists of a sequence of equal-sized pages which are referenced and formatted by various components of the RSS. The RSS maintains a page map for each segment which is used to map each segment page to its location on disk. At RSI, segments are identified by a numeric segment identifier. Pages are identified by page number within segment. Pages are never directly referenced in SEQUEL.

2. Files and Records

Each base table is represented as a stored file. A stored file is identified at the RSI by a numeric identifier called as RID. In other words, a RID identifies a stored file. The RDS is responsible for mapping SEQUEL table-names to RDIs. Records in the stored file represent rows of the table. Each record is stored as byte string. The byte string

consists of a prefix, (containing control information, such as the RID of the containing file), followed by the stored representation of each field in the record. Like segments and files , individual tuples have their own numeric identifier, called a TID. The TID for a tuple consists of two parts: page number of the page containing tuple, and a byte offset from the bottom of the page identifying a slot that contains, in turn, the byte offset of the tuple from top of the page. Operators are available to INSERT and DELETE single tuples, and to FETCH and UPDATE any combination of fields in a tuple.

3. Images and Links

An image in the RSS is a logical reordering of an n-ary relation with respect to values in one or more sort fields. Images combined with scans provide the ability to scan relations along a value ordering for low level support of simple views. An image provides associative access capability. The RDS can rapidly fetch a tuple from an image by keying on the sort field values. A new image can be defined at any time on any combination of fields in a relation. Each of the fields may be specified as ascending or descending. An image can also be dropped at any time. The RSS maintains each image through the use of multipages index structure. A new page can be added to an index when needed as long as one of the pages within the segment is marked as available. The pages for a given index are organized into a balanced hierarchic structure. Each page is a node within the hierarchy and contains an ordered sequence of index entries.

A link in the RSS is an access path which is used to connect tuples in one or more relations. The RDS determines which tuples will be on the link and determines their relative position by using explicitly the CONNECT and

DISCONNECT operations. The RSS maintains internal pointers so that newly connected tuples are linked to previous and next twins, and previous and next twins are linked to each other when a tuple is disconnected.

4. Transaction Management

A transaction at the RSS is a sequence of RSI calls in behalf of one user. In general, an RSS transaction consists of those calls generated by the RDS to execute all RDI operators in a single System R transaction, including the calls required to perform such RDS internal functions as authorization, catalog access, and integrity checking. An RSS transaction is marked by the START-TRANS and END-TRANS operators. A transaction save point is marked as the SAVE-TRANS operator, which returns a save point number of subsequent reference. In general, a save point may be generated by any of the layers above the RSS. An RDI user may mark a save point at a convenient place in this transaction in order to handle backout and retry. The RDS may mark a save point for each new set oriented SEQUEL expression. Transaction recovery occurs when the RDS or Monitor issues the RESTORE-TRANS operator, which has a save point number as its input parameter, or when the RSS initiates the procedure to handle deadlock. The transaction recovery function is supported through the maintenance of the time ordered lists of log entries, which record information about each change to recoverable data. Those changes include all the tuple and image modifications caused by INSERT,DELETE, and UPDATE operations and all the link modifications caused by CONNECT and DISCONNECT operations.

5. Concurrency Control

Since System R is a concurrent user system, locking techniques must be employed to solve various synchronization

problems, both at the logical level of objects like relations and tuples and at the physical level of pages. At the logical level, such classic situations as the "lost update" problem must be handled to insure that two concurrent transactions do not read the same value and then try to write back an incremented value. If these transactions are not synchronized, the second update will overwrite the first, and the effect of the increment will be lost. At the physical level of pages, locking techniques are required to insure that internal components of the RSS give correct results.

6. Locking

One basic decision in establishing System R was to handle both logical and physical locking requirements within the RSS, rather than splitting the functions across the RDS and RSS subsystem. Physical locking is handled by setting and holding locks on one or more pages during the execution of a single RSI operation. Logical locking is handled by setting locks on such objects as sequence, relations, tuple identifiers (TIDs), and key value intervals and holding them until they are explicitly released or to the end of the transaction. Another basic decision in formulating System R was to automate all of the locking functions, both logical and physical, so that a user can access shared data and delegate some or all lock protocols to the system.

In order to provide reasonable performance for a wide spectrum of user requirements, the RSS supports multilevels of consistency which control the isolation of a user from the actions of the other concurrent users [Ref.2]. When a transaction is started at the RSI, one of three consistency levels must be specified. Different consistency levels may be chosen by different concurrent transactions. For all of these levels, the RSS guarantees that any data

modified by the transaction is not modified by any other until the given transaction ends. The differences in consistency levels occur during read operations. Level-1 consistency offers the least isolation from the other users, but causes the lowest overhead and lock attention. With this level, dirty data may be accessed, and one may read different values for the same data item during the same transaction. In level-2, the user is assured that every item read is clean. However, no guarantee is made that subsequent access to the same item will yield the same values or that associative access will yield the same item. For the highest consistency level (which is level-3) the user sees the logical equivalent of a single user system. Every item read is clean, and subsequent reads yield the same values, subject to updates by the given user. Level-3 consistency eliminates the problem of lost updates and also guarantees that one can read a logically consistent version of any collection of tuples, since other transactions are logically serialized with the given one.

The RSS components set locks automatically in order to guarantee the logical functions of these various consistency levels. The RSS employs a single lock mechanism to synchronize access to all objects. This synchronization is handled by a set of procedures in every activation of the RSS, which maintains a collection of queue structures called GATES in shared, read write memory. An internal request to lock on an object has several parameters: object name, lock mode, and indication of lock duration. There are several factors which will effect the choice of lock duration such as the type of action requested by the user and consistency level of the transaction. Data items can be locked at various granularities to insure that various applications run efficiently. Lock on a single tuple will be effective for transactions which access small amounts of data. Locks

on entire relations or even entire segments will be more reasonable for transactions which cause the RDS to access large amounts of data. For accomplishing these situations, a dynamic lock hierarchy protocol has been developed so that a small number of locks can be used to lock both few and many objects.

7. Deadlock

Since locks are requested dynamically, it is possible for two or more concurrent activations of the RSS to deadlock. The RSS has been designed to check for deadlock situations when requests are blocked and to select one or more victims for backout if deadlock is detected. The detection is done by the Monitor on a periodic basis by looking for cycles in a user-user matrix. The selection of victim is based on the relative ages of transactions in each deadlock cycle as well as on the duration of the locks. In general the RSS selects the youngest transaction whose lock is of short duration, since the partially completed call can easily be undone. If none of the locks in the cycle are of short duration, the youngest transaction is chosen. This transaction is then backout to the save point preceding offending lock request, using the transaction recovery scheme.

IX. IMPLEMENTATION BY USING ORACLE

A. INTRODUCTION

The CRACLE Relational Database Management System is a computer program that manages pieces of data stored in a computer. ORACLE allows access to this data by providing sets cf commands that tell the computer what to do. These commands are in a language that is called SQL. SQL has several facilities fcr data manipulation. Some of them will be used for the Inventory Database.

All data in ORACLE are stored as tables. Tables are made up of columns and rows. The SUPPLIER table shown below has four cclumns (SUPP_NAME, COUNTRY, ADDRESS, and SHIP_TYPE) and four rows. A rcw is made up of fields. Each field contains a data value stored where a column and row meet. For example, the first row in the SUPPLIER table has the data value ITT stored in its SUPP_NAME field, the data value USA stored in its CCUNTRY field, PO.BOX.9 stored in its ADDRESS field, and the data value S.F stored in its CITY field. A database can contains many tables. ORACLE allows the creation of as many tables as needed. All the tables stored in ORACLE make up the database.

We can create a table using the CREATE TABLE command. The ccmmand that creates the PART_IDENTIFICATION table is as follows:

```
UFI> r
  1  create table part_identification
  2         ( nsn_no char(14),
  3           tot_qty_on_hand number(6),
  4           max_auth_qty_hand number(6),
  5*          sum_of_used_unit number(6) )

Table created.
```

In the CREATE TABLE command we name the table PART_IDENTIFICATION and the columns of the table (NSN_NO, TOT_QTY_ON_HAND, MAX_AUTH_QTY_HAND, SUM_OF_USED_UNIT). We specify if the column is to contain only numeric values (NUMBER) or character (both numbers and letters) values (CHAR). We also specify the maximum length of the value that can be stored in the column. For example, no NSN_NO can be longer than 14 characters- nsn_no char(14).

After a table is created, rows can be entered into the table using the INSERT command . The following command is used to enter the first row into the PART_IDENTIFICATION table.

```
UFI> insert into part+identification
  2  values ('1342-241-4111',15000,20000,30000);

1 record created.
```

In the INSERT command we name the table PART_IDENTIFICATION into which the row is to be inserted and list the data values that go into each column.

In a similar manner using the CREATE TABLE command, all tables in the inventory database are created and using the INSERT command all data are inserted into tables. The final version of the tables are shown below.

PART_IDENTIFICATION

```
UFI> select *
  2  from part+identification;

NSN+NO          TOT+QTY+ON+HAND MAX+AUTH+QTY+HAND SUM+OF+USED+UNIT
--------------- --------------- ----------------- ----------------
1342-241-4111             15000             20000            30000
2421-311-4111             10000             15000            20000
2451-312-4115              5000             10000            90000
5111-111-1511              25000             10000            15000
2511-511-4511             10000             15000            20000
1015-512-5112              2000              4000             1500
7511-632-8332             15000             25000           125000

7 records selected.
```

DCCUMENT_IDENTIFICATION

```
UFI> select *
  2  from document+identification;

NSN+NO          DOCU SUPP
--------------- ---- ----
1342-241-4111 tom1 itt
2421-311-4115 tom1 itt
2451-312-4115 tom2 asal
5111-111-1511 tom1 asal
2551-511-5411 tom1 dec
1511-215-5111 tom5 ibm
1015-512-5112 tom2 dec
7511-632-8332 tom3 asal

8 records selected.
```

UNIT_INVENTORY

```
UFI> select *
  2  from unit+inventory;

UNIT+C NSN+NO+USE     QTY+ON+HAND USED+AMOUNT REQ+AMOUNT
------ -------------- ----------- ----------- -----------
1base  1342-241-4111         7500       30000        2500
1base  2421-311-4115         2000       10000        3000
2base  2412-311-4115         3000       10000        2000
2base  2451-312-4115         1250        4000         750
3base  2451-312-4117         1250        4000         750
3base  2511-511-4511         5000       20000        2000
4base  1511-215-5111         6250        7500        3200
5base  1511-215-5111         5000       20000        2000
6base  7511-632-8332          600        1000         400
7base  1015-512-5112          400         500         200
8base  1015-512-5112         1250        5000        3250
9base  5111-111-5111         7500       25000       10000
```

UNIT_ID

```
UFI> select *
  2  from unit_id;

UNIT_C SUPERI LOCATION
------ ------ --------
1base  2taf   13N20E
2base  2taf   25N30E
3base  3taf   21N32E
4base  4taf   36N21E
5base  4taf   40N22E
6base  1taf   31N25E
7base  1taf   37N25E
8base  2taf   38N25E
9base  3taf   41N21E

9 records selected.
```

PART_ORDER

```
UFI> select *
  2  from part_order;

NSN_NO      SUPP_N DATES       AMOUNT SHIP
----------- ------ ------ ---------- ----
1342-241-4111 itt    031685       2500 air
2451-312-4115 asal   012585       5000 sea
5111-111-1511 itt    031685       3250 air
1511-215-5111 dec    011685       5250 sea
2511-512-5111 asal   031785       5000 sea
2015-512-5112 itt    042585        600 air
2015-632-8332 iom    052785      10000 air

7 records selected.
```

DEPOT_STOCK_LEVEL

```
UFI> select *
  2  from depot+stock+level;

DEPO+ STOCK+AMOUNT SUPP NSN+NO+REGIST
----- ------------ ---- -------------
depo1          7500 itt  1342-241-411
depo2          5000 asal 2421-311-4115
depo3          2500 ibm  2451-312-4115
depo4          5000 dec  2511-312-4115
depo5          1000 ibm  1015-512-5112
depo6          7500 dec  7511-682-8332
depo7         11250 ibm  1511-215-5111
depo8          1250 asal 5111-111-5111



8 records selected.
```

SUPPLIER

```
UFI> select *
  2  from supplier;

SUPP COUNTR ADDRESS              CITY
---- ------ -------------------- -----------
itt  usa    po.box.9             S.F
ibm  usa    po.box.12            L.A
dec  usa    po.box.11            N.Y
asal tur    po.box.7             izmir
```

B. SAMPLE QUERIES

1. List national stock number of the items for which the quantity on hand equal 10000.

```
UFI> select nsn_no
  2  from part_identification
  3  where tot_qty_on_hand = 10000;

NSN_NO
--------------
2421-311-4111
2511-511-4511
```

2. Display nsn_no which is in the 1base and required amount greater than 2500.

```
UFI> select nsn_no_use
  2  from unit_inventory
  3  where unit_code = '1base'
  4  and  req_amount > 2500;

NSN_NO_USE
-------------
2421-311-4115
```

3. List all bases which are under command of 2taf.

```
UFI> select unit_code
  2  from unit_id
  3  where superior_comm = '2taf' ;

UNIT_C
------
1base
2base
8base
```

4. List all locations of bases which are under command of 1taf.

```
UFI> select unit_code,location
  2  from unit_id
  3  where superior_comm = '1taf' ;

UNIT_C LOCATION
------ --------
6base  31N25E
7base  37N25E
```

5. List all suppliers names and their addresses in the USA.

```
UFI> select suppname,address,city
  2  from supplier
  3  where country = 'usa' ;

SUPP ADDRESS              CITY
---- -------------------- -----------
itt  po.box.9             S.F
ibm  po.box.12            L.A
dec  po.box.11            N.Y
```

6. Find total quantity on hand, document, and supplier name for items for which the sum of used amount is greater than 1000.

```
UFI> select tot_qty_on_hand,document,suppname
  2  from part_identification,document_identification
  3  where sum_of_use_unit > 1000
  4  and part_identification.nsn_no = document_identification.nsn_no;

TOT_QTY_ON_HAND DOCU SUPP
--------------- ---- ----
          15000 tom1 itt
           5000 tom2 asal
           2000 tom2 dec
          15000 tom3 asal
```

7. Find total used amount for 1base.

```
UFI> select sum(used_amount)
  2  from unit_inventory
  3  where unit_code = '1base' ;

 SUM(USED_AMOUNT)
-----------------
            40000
```

8. Find nsn_no and total quantity on hand in descending order by tot_qty_on_hand for items for which the maximum authorized quantity on hand is greater than 1500.

```
UFI> select nsn_no,tot_qty_on_hand
  2  from part_identification
  3  where max_auth_qty_hand > 1500
  4  order by tot_qty_on_hand desc ;

NSN_NO          TOT_QTY_ON_HAND
--------------- ---------------
5111-111-15111            25000
1342-241-4111             15000
7511-632-8332             15000
2421-311-4111             10000
2511-511-4511             10000
2451-312-4115              5000
1015-512-5112              2000

7 records selected.
```

9. Display total quantity on hand and sum of used amount for items supplied by ASAL.

```
UFI> select tot_qty_on_hand,sum_of_used_unit
  2  from part_identification
  3  where nsn_no in
  4        ( select nsn_no
  5          from document_identification
  6          where supp_name = 'asal');

TOT_QTY_ON_HAND SUM_OF_USED_UNIT
--------------- ----------------
           5000            90000
          15000           125000
```

10. Find order amount, dates, and shipment type for items which are in the 1BASE inventory.

```
UFI> select amount,dates,ship_type
  2  from part_order
  3  where nsn_no in
  4        ( select nsn_no_use
  5          from unit_inventory
  6          where unit_code = '1base');

    AMOUNT DATES  SHIP
---------- ------ ----
      2500 031685 air
```

11. Find required amount and order amount for items in the 2BASE inventory.

```
UFI> select req_amount,amount
  2  from unit_inventory,part_order
  3  where unit_inventory.unit_code = '2base'
  4  and unit_inventory.nsn_no_use = part_order.nsn_no ;

REQ_AMOUNT     AMOUNT
---------- ----------
       750       5000
```

12. Find total quantity on hand and maximum authorize amount on hand for items for which nsn_nc is 1015-512-5112.

```
UFI> select tot_qty_on_hand,max_auth_qty_hand
  2  from part_identification
  3  where nsn_no = '1015-512-5112' ;

TOT_QTY_ON_HAND MAX_AUTH_QTY_HAND
--------------- -----------------
           2000              4000
```

X. CONCLUSIONS AND RECOMMENDATIONS

An inventory database system is complex and important. In order to effectively command and control the inventory of an Air Force, the commander must know the status of his resources which will present the state of operational readiness of the Air Force. It is difficult to obtain accurate information from the inventory system by using the manual systems. The database management system must be used in the inventory systems in order to increase end-user productivity, decrease staff,and enable work to be done more efficiently.

The complex task of a logical database design for a relational database management system can be greatly simplied by use of the Semantic Data Model. SDM is a high-level semantics-based database description and structuring formalism for the database and enhances usability of the database system. Using the output of SDM in the Inventory Database, the records are rearranged in crder to fit a relational model. ORACLE DBMS was used for implementation. Functionality of ORACLE DBMS is very high and provides User Friendly Interface (UFI). It is easy to use fcr all potential users.

Finally, database machines are being developed in universities and research laboratories. It is obvious that a great deal of effcrt is being devoted to developing, studyirg and analyzing database computers. These efforts will result in quality hardware and software for all potential users of relational database management systems.

LIST OF REFERENCES

1. C. J. Date, An Introduction to Database System, IBM Corporation, 1982 , pages 203-234.

2. Atre, S. , Database: Structured Techniques for Design Performance and Management with case Studies, Business Data Processing: A Wiley series, 1980, pages 231-236.

3. Ronald G.Ross, Database Systems: Design Implementation and Management, AMOCOM, 1978

4. David Kronke, Database Processing: Fundamentals Design Implementation , Science Research Associates, Inc. 1983,

5. Jayanta Banarjee and David K. Hsiao, DBC Software Requirement for Supporting Relational Database, November 1977,

6. M. M. Astrahan et al, "System R: Relational Approach to Database Management", ACM Transactions on Database Systems, No.2, June 1976,

7. M. M. Astrahan et el, "System R, A Relational Database Management System", IEEE Computer Society: Computer, 12 No.5, May 1979,

8. R. F. Boyce and D. D.Chamberlain, "Using a Structured English Query Language as a Data Definition Facility", IBM Research Report RJ1318, December 1973,

9. E.F.Codd,"Recent Investigations Into Relational Database System", ACM Pasific Conference, San Francisco, April 1975,

10. Toby. J. Teorey and James P. Fry, Design of Database Structure, 1982

11. Hawryszkiewycz, I.T, Database Analysis and Design Science Research Associate, Inc.1984,pages 123-145.

12. Atre, S., Database: Structure Techniques for Design Performance and management, 1980,

13. M.Hammer and D. McLoed, "Database description with SDM: A Semantic Database Model", ACM Transaction on database system, Vol.6 No.3, September 1981,pages 351-386.

INITIAL DISTRIBUTION LIST

No.		Copies
1.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5100	2
2.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93943-5100	1
3.	Professor S. H. Parry, Code 55Py Department of Operation Research Naval Postgraduate School Monterey, California 93943-5100	2
4.	Department of Logistics Turkish Air Force Headquarters Bakanliklar, Ankara, TURKEY	2
5.	Osman SARI Zafer Mahallesi, Kaymakci Odemis, Izmir, TURKEY	9
6.	Division of Education Department of Personal Turkish Air Force Headquarters Bakanliklar, Ankara, TURKEY	2
7.	Ugur OZKAN Hukumet caddesi Sunullah Bey Apt. No: 7/4 Kayseri, TURKEY	1
8.	Hava Harp Okulu Komutanligi Kutuphane Yesilyurt,Istanbul, TURKEY	1
9.	Hava Harp Academisi Komutanligi Kutuphane Ayazaga,Istanbul, TURKEY	1
10.	EIBM Komutanligi CEIM Mudurlugu Eskisehir, TURKEY	1
11.	AIBM Komutanligi CEIM Mudurlugu Etimesgut, Ankara, TURKEY	1

12. ODTU Bilgisayar Muhendisligi 1
Dekanligi
Ankara, TURKEY

13. ITU Bilgisayar Muhendisligi 1
Dekanligi
Gumussuyu,Istanbul, TURKEY

14. Bogazici Universitesi 1
Bilgisayar Muhendisligi
Dekanligi
Rumelikavagi,Istanbul, TURKEY

15. Hava Egitim Komutanligi 2
Egitim Sb. Md.
Guzelyali,Izmir, TURKEY

16. Defense Technical Information Center 2
Cameron Station
Alexandria, Virginia 22304-6145